3-24-2016

POCO-MOEA: Using Evolutionary Algorithms to Solve the Controller Placement Problem

Scott I. Harned

POCO-MOEA: USING EVOLUTIONARY ALGORITHMS TO SOLVE THE CONTROLLER PLACEMENT PROBLEM

THESIS

Scott I. Harned, Captain, USAF

AFIT-ENG-MS-16-M-021

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-16-M-021

POCO-MOEA: USING EVOLUTIONARY ALGORITHMS TO SOLVE THE CONTROLLER PLACEMENT PROBLEM

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Scott I. Harned, B.S.E.E.

Captain, USAF

24 March 2016

AFIT-ENG-MS-16-M-021

POCO-MOEA: USING EVOLUTIONARY ALGORITHMS TO SOLVE THE CONTROLLER PLACEMENT PROBLEM

THESIS

Scott I. Harned, B.S.E.E.
Captain, USAF

Committee Membership:

Barry E. Mullins, Ph.D.
Chair

Timothy H. Lacey, Ph.D.
Member

Michael R. Grimaila, Ph.D., CISM, CISSP
Member

Gary B. Lamont, Ph.D.
Member

AFIT-ENG-MS-16-M-021

Abstract

One of the central tenets of a Software Defined Network (SDN) is the use of controllers, which are responsible for managing how traffic flows through switches, routers, and other data-passing devices on a computer network. Most modern SDNs use multiple controllers to divide responsibility for network switches while keeping communication latency low. A problem that has emerged since approximately 2011 is the decision of where to place these controllers to create the most 'optimum' network. This is known as the Controller Placement Problem (CPP). Such a decision is subject to multiple and sometimes conflicting goals, making the CPP a type of Multi-Objective Problem (MOP).

The Controller Placement Problem is NP-Hard. This means finding the 'optimum' solution can become a time-intensive process as network size increases. Multiple algorithms exist to solve MOPs using shortcut (or 'heuristic') methods which can produce a 'near-optimal' solution in times much shorter than those necessary to guarantee an 'optimal' solution. One popular class of algorithms is known as Evolutionary Algorithms (EAs); EAs designed to solve Multi-Objective problems are called Multi-Objective Evolutionary Algorithms (MOEAs). While many MOEAs exist, their application to the Controller Placement Problem is not well explored. The theory of this thesis is that an MOEA can produce solutions to the Controller Placement Problem which are 'nearly optimal' while keeping execution time low compared to an exhaustive 'optimal' search. This research extends a network modeling tool called the Pareto Optimal Controller Placement (POCO) Framework with custom designed MOEA, called POCO-MOEA. A series of full-factorial experiments is designed and executed to gather data on POCO-MOEA performance to a series of

model networks. The algorithm's behavior is then evaluated and compared to exhaustive search through five metrics; fraction of solution space size, average distance between pareto fronts (δ_1), worst-case distance between pareto fronts (δ_2), relative hypervolume (hyp_{rel}), and relative execution time (b_{rel}).

Results show that performance is dependent on the size of the network, the topology of the network, and the parameters chosen for POCO-MOEA. In general, performance for POCO-MOEA improves as the size of the network increases. Given a large network (60+ nodes), POCO-MOEA can achieve within 0.4% of δ_1, 3% of δ_2, and 6% of hyp_{rel} while still being 500 times faster than exhaustive search. This research demonstrates and adds a valuable tool to the methods of determining optimal device placement for an SDN while providing steps to using MOEAs in real SDN applications.

Table of Contents

Page

Page

List of Figures

List of Tables

Table Page

List of Abbreviations

Abbreviation		Page
SA	Simulated Annealing	32
MaOEA	Many Objective Evolutionary Algorithm	38
SUT	System Under Test	57
HV	Hypervolume	61
GD	Generational Distance	61
HDF5	Hierarchical Data Format, Version 5	68
GECCO	Genetic and Evolutionary Computation Conference	99
ACO	Ant Colony Optimization	99
EA	Evolutionary Algorithm	99
TSP	Traveling Salesman Problem	99

List of Algorithms

POCO-MOEA: USING EVOLUTIONARY ALGORITHMS TO SOLVE THE CONTROLLER PLACEMENT PROBLEM

I. Introduction

1.1 History

One of the greatest roadblocks in communications technology innovation over the last 10 years has been network management. Network hardware has become progressively more powerful, yet traditional Internet Protocol (IP) networks are still controlled by switches and routers that have to be individually configured, updated and maintained [1]. This decentralized method of network administration creates an egregious amount of work for maintainers. Network administrators not only must be trained on multiple platforms for a single class of device. They must also do so for a wide range of device classes (routers, switches, firewalls, proxies, etc.) Each of these devices can have a significant impact on the behavior of a network, where a single programming error can result in a disruption of service for a large portion of the user-base. While there were a few experiments in the late 1990s to implement a software defined architecture [2], none of these efforts gained much traction.

In 2005, Greenberg *et. al.* diagrammed a new 'architecture intent' [3] known as the '4D' architecture. It divided networks into four 'planes' of management: decision, dissemination, discovery, and data. It was the first paper to successfully advance the idea of what would become Software Defined Networking (SDN).

1.2 The Problem

This research focuses on one component of the SDN architecture, the network controller. This device is responsible for management of traffic paths on an SDN; some of the history and types of this device is covered in Chapter II. The central portion of this thesis covers a new problem that has arisen with the use of distributed SDN controllers; the decision of where to place said controllers on a network. This problem is known as the Controller Placement Problem (CPP), which is a subset of the *Plant, Warehouse or Facility Location Problem* [4, 5]. Network administrators have a limited number of controllers available to place, as well as a limited number of locations which can support those controllers. The ultimate choice of where these controllers are placed can have a significant impact on the ultimate performance and behavior of the network. There are a multitude of performance factors which can change depending on characteristics of a network one wishes to control, such as network resilience, inter-controller latency, controller-to-non-controller latency, network cost, and others. The fact that most of these values can change over time can add an additional wrinkle to decision making. Because of the multiple, sometimes conflicting, goals involved in solving these types of problems, they are referred to as a Multi-Objective Problem (MOP).

This naturally leads to the desire to find the best (or most 'optimum') placement of these controllers. This thesis discusses the difficulties inherent in determining this most optimum placement. However, difficulties arise in both the computational complexity and the definition of the term 'optimum' for problems of this type.

The CPP, as a subset of the *Plant, Warehouse or Facility Location Problem*, is known to be of the Non-Polynomial (NP)-Hard class of complexity [6, 7], meaning that the computational time required to find the 'optimum' solution of the problem increases exponentially with the size of the network. The reason for this time increase

is that in order to guarantee that the 'optimum' solution is found, a program must iterate through every possible combination of controllers to available controller locations to guarantee all truly 'optimum' solutions have been found. For the CPP, this means finding all solutions in which out of n possible network locations, k are chosen to be controllers (an n-choose-k problem). In most cases a truly 'optimum' solution does not even exist, and instead a single answer must be chosen from a set of solutions which represent trade-offs between optimization criteria. As previous research has shown, such a problem can extend from a few seconds of computation time for a small network to several hours for a large one [8].

1.3 Heuristics

In order to combat this type of problem complexity, many algorithms have been developed which can produce a 'near-optimum' set of solutions in a much more reasonable length of time compared to exhaustive algorithms which produce an 'optimum' solution. They do this by limiting their behavior to exploring only a small fraction of all possible solutions for a problem. The performance of these heuristics is determined by predetermined objectives that one seeks to optimize. These types of algorithms are referred to as 'heuristics'. While these programs cannot guarantee finding an 'optimum' solution, these types of programs have proven over time they can come relatively close to 'optimum' solutions, suitable for most types of MOPs.

While there are many types of heuristics in existence, the application of heuristics to the CPP is not well explored. Two heuristics have been developed specifically to provide more rapid solutions, with varying degrees of trade-offs between speed, complexity and performance. These heuristics exist within a program made to solve the CPP called the Pareto Optimal Controller Placement (POCO) program. This thesis explores the development and evaluation of a third heuristic based on a Multi

Objective Evolutionary Algorithm (MOEA). This heuristic is modeled after one of the most well known MOEAs, the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [9]. As this heuristic is built specifically for POCO, it is given the name POCO-MOEA.

1.4 Hypothesis and Methodology

The hypothesis of this thesis is the implementation of POCO-MOEA can produce a set of solutions which are a fraction of the size of an exhaustive search, with values based on a set of objective functions that are nearly equal to those of exhaustive search. In addition, the MOEA can produce these nearly optimal solutions at least 10 times faster compared to exhaustive search. Within POCO-MOEA itself, it is believed that the initial input parameters for POCO-MOEA can have a significant impact on the output. The goal of the thesis is to determine whether POCO-MOEA can meet the targets and just how much parameter inputs affect the heuristic.

In order to determine how well POCO-MOEA performs compared to exhaustive search, a full-factorial experiment is built using the POCO framework. Four network models retrieved from the Internet Topology Zoo [10] are used as the testing environments for the data runs. The experiment relies on adjusting between these four network models, three different solution set sizes and four different iteration loop counts (called generations) of POCO-MOEA.

1.5 Assumptions

There are a couple of fundamental assumptions and limitations that are used for this series of experiments to assist in simplifying the development of this thesis:

- The use of POCO framework model as opposed to a live network.
- Every node on the network is selectable as a location for a controller.

- The network is static; devices are never added or removed, and behavior never changes during the course of testing.
- The network is fully reliable; that is, there is no chance of a network node, controller or link failure to interrupt performance.
- POCO-MOEA is only compared to exhaustive search; performance comparisons to other heuristics are not examined.
- Code efficiency is not examined; while optimizations may exist for POCO-MOEA which could impact metrics (specifically b_{rel} - see Chapter IV), they are not examined in this thesis.
- Several different networks are used for experiments; Each of the networks has a different size and layout, which has an impact on the resulting metrics from experiment runs. The overall effect of network characteristics on experimental results is not thoroughly explored, but their impact on certain aspects of metric behavior are roughly examined.

The benefits of this thesis is that it adds a new tool for solving the CPP. As each type of heuristic tends to perform with different efficiency depending on the type of problem being solved, it is beneficial to test as many algorithms as possible on each problem to determine which is most suitable [6]. Explorations in this field also provide a stepping stone for future researchers to extend into more realistic evaluations of the performance characteristics of SDN controllers, such as the use of simulators or live networks in place of models.

1.6 Thesis Layout

The rest of this thesis proceeds in the following manner. Chapter II describes the history, architecture and structure of SDN. In addition, some of the current popular centralized and distributed controller software programs are also discussed. Next,

this work analyzes several research documents which provide algorithms to solve the CPP through either a single objective or MOP format. Chapter III provides an explanation of the inner workings of MOEAs as well as a formal algorithm specification of POCO-MOEA. Chapter IV explains the parameters that have been chosen for the full-factorial analysis experiment of POCO-MOEA. The metrics used to analyze the behavior of the heuristic are also described. Chapter V provides the factorial experiment data and denotes some of the expected (as well as unexpected) results. Chapter VI provides conclusions based on the gathered data, introduces theories for some of the unexpected heuristic behavior and postulates possible directions for future research. For additional details, Appendix A provides more detail on how the POCO Framework is structured, while Appendix B provides snippets of early experimental data which led to the decision for the parameters used in the actual factorial experiments performed in Chapter IV.

II. Background

2.1 Introduction - Software Defined Networking

In traditional networking, data transported over a network and the information for controlling the routing of the data are carried over the same communication channels. Moreover, the decisions of how to route packets and the actual routing are calculated by the same individual devices. This results in fragmented and inconsistent network management. While this could potentially make for a robust network, it becomes at the same time difficult to manage and cumbersome to maintain/update.

In contrast to the traditional network format, an SDN makes the network easier to manage as it promotes flexibility. While there are differing opinions as to the formal definition of SDN, there are several commonly accepted components. A traditional network passes network traffic and the instructions for how to route that traffic together, often within the same data packets. This single 'plane' of information is managed by the switches, routers, firewalls, anti-virus tools – in short, all of the functions related to traffic monitoring, security and route determination. In contrast, the paradigm of SDN separates these two parts into separate 'planes' of information; these are known as the 'data' plane and the 'control' plane of the network [11]. In an SDN, nearly all of the responsibility for route determination is removed from the aforementioned devices, and they become strictly the domain of data routing. The data plane is then only responsible for forwarding data across the network. Devices on the control plane assume responsibility for determining and implementing the routing paths for data that moves across a network. These control plane devices then create sets of rules which are then installed on the data plane. This control plane is managed by new specialized devices on the network called 'controllers'. Most often, these controllers exist as virtualized software on a server [12, 13, 14, 15, 16]. Several

of these tools are identified in Section 2.4. Additional tools such as anti-virus and firewall are outside the scope of this research, however they can exist in an SDN in a somewhat modified format from a traditional network. This is slightly expanded on later in this chapter.

Since the initial introduction of the data/control plane model of SDN, it has progressively evolved to accommodate more features and capabilities, enabling additional services to be administered via the control layer (such as firewall and anti-virus). Today, SDN is commonly accepted as a three tier, five layer system, characterized in [11] and illustrated in Figure 1. A more detailed illustration of interactions between the planes is given in Figure 2.

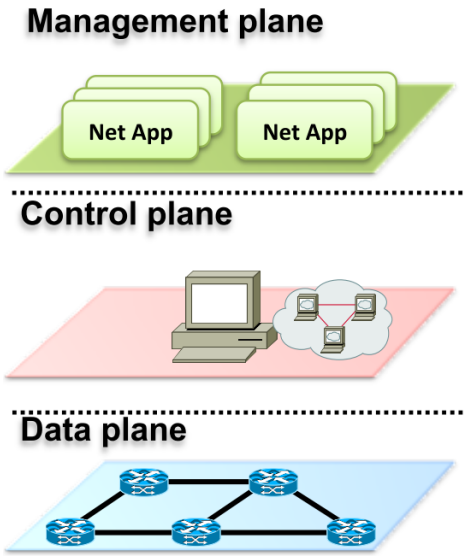

Figure 1. Three-Tiered SDN Illustration [11]

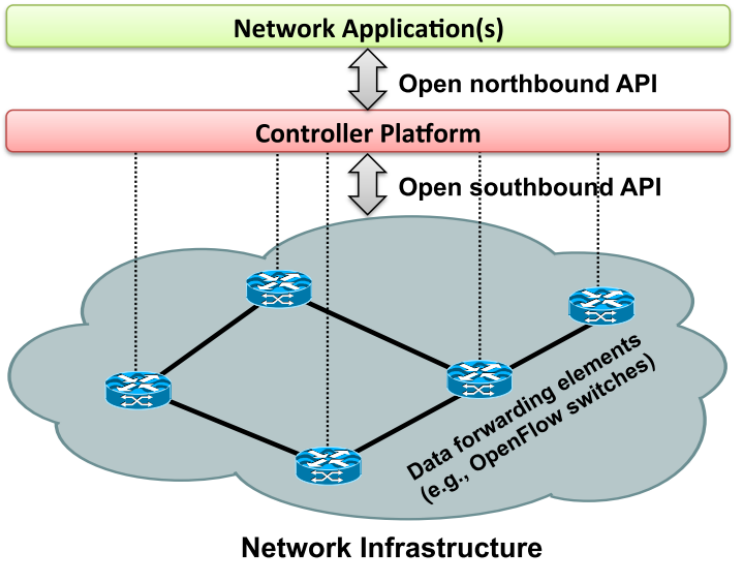

Figure 2. A Simplified View of SDN Architecture [11]

The first tier is illustrated in the bottom third of Figure 1, called the 'data plane'. It is composed of all switches responsible for passing actual traffic data, also known as the Network Infrastructure (see the bottom portion of Figure 2). These switches are simpleminded, containing only data tables for purposes of packet routing. These 'data tables' are lists of information designed to categorize packets for deciding how to route said packets to their destinations. This allows data plane switches to work more efficiently by devoting all available computation power to the purpose of packet forwarding. [11].

The 'control plane' (middle portion of Figure 1) is the second tier, comprised of device(s) responsible for deciding how the 'data tables' for every switch is populated. This layer is also alternatively defined as the Controller Platform in the middle portion of Figure 2. This layer can be made up of one device or several, depending on the size of the network. Control plane devices are responsible for managing multiple and various data plane devices. This allows a network administrator to set policy for altering routing behavior on multiple devices from a single physical or logical location,

allowing greater speed and flexibility in modification of a network's routing behavior.

In order for the data and control planes to communicate with each other, an intermediary layer called the 'southbound Application Program Interface (API)' is used to foster communication (Figure 2). This is where the OpenFlow protocol resides; it provides a language that the two planes use to communicate. The control plane devices provide updated data routing tables (called 'flow tables', see Section 2.2) to the data plane, while the data plane provides updated network performance and behavior statistics to help the control plane make intelligent tables. While there are alternatives to OpenFlow available, it has become the *de facto* standard [11].

The third tier, the newest, is the 'management plane' (top third of Figure 1). The emergence of a management plane has come as part of a larger shift towards the concept of Network Functions Virtualization (NFV) [17]. As part of the move toward NFV, more traditional network management tools (firewalls, anti-virus, load-balancing, wireless, traffic engineering, access control, security) have become virtualized applications rather than software sitting on dedicated hardware devices attached to a network. Because many of these applications can have an impact on how packets traverse a network, these applications sit above the controllers on the control plane (top third of Figure 2). In this way these tools can influence the way the control layer makes decisions on how traffic is routed, while the control layer can provide applications with information about the network to make better decisions [11].

The last layer, the 'northbound API', is the interface between the management and control planes (between Network Applications and Controller Platform of Figure 2). Similar to the 'southbound API' between the data and control planes. This is currently the least standardized layer of the SDN architecture, as applications have differing requirements on what information needs to be communicated to/from/between the controllers, and the format in which said information is transferred [11].

2.2 OpenFlow

While push toward SDN began in earnest in 2005, it was not until the introduction of OpenFlow in 2008 that a simplified method for actually implementing a software defined architecture became achievable [18]. There are currently two versions of the protocol in wide usage throughout most switches today, 1.0 (published in 2009) [19] and 1.3 (published in 2012) [20]. Version 1.5 was draft approved in December 2014 and is still being formally ratified [21]. The most recent version of the protocol as of this writing is v1.5.1, published in March 2015 [22].

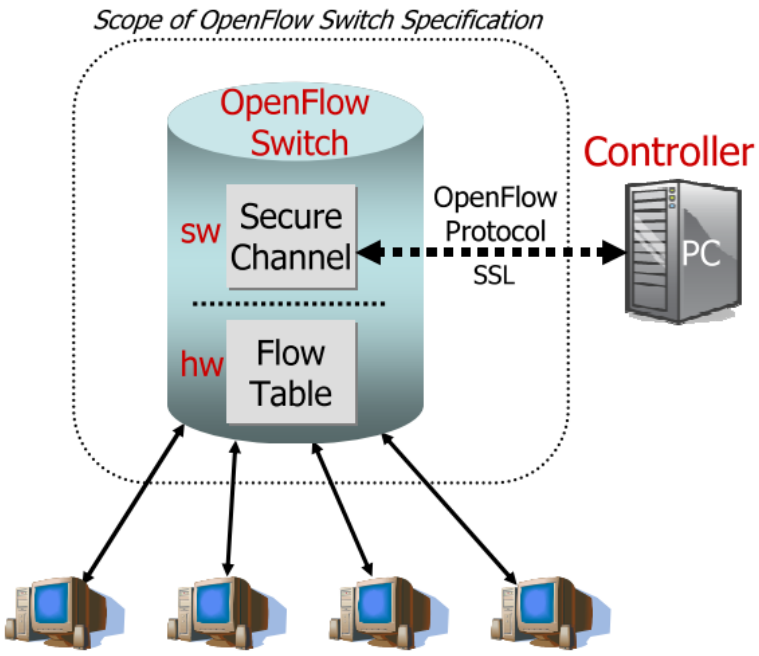

Figure 3. A Simplified View of OpenFlow Switch Operation [18]

Figure 3 provides a simplified description of how OpenFlow is used in a network. An OpenFlow switch can be any switch coded to run the OpenFlow protocol. The controller on the right of the figure sends instructions to the OpenFlow switch via a Secure Socket Layer (SSL) channel. The OpenFlow-enabled switch maintains these instructions as databases for processing and forwarding packets. These are the 'flow tables' mentioned in Section 2.1. A single switch can contain multiple flow tables,

which are traversed in order for each packet to determine its destination. A match in one flow table can influence which other switches, and therefore flow tables, a packet passes through. The OpenFlow protocol requires that a 'table-miss' flow be placed at the bottom of every flow table; if a packet cannot find a match for any flow currently on the switch, it hits the 'table-miss' entry by default. The network administrator can configure the protocol to then either drop the packet, or forward it to the switch's controller to generate a new flow. The switch then uses this table to route packets to the appropriate devices, represented by the desktop PCs at the bottom of the figure. These can be any device which can receive packets (workstations, servers, mobile devices, etc.) Because the switch is not making decisions on how to route traffic to the proper destinations, it can focus all available processing power on the actual routing.

While most people today associate SDN with OpenFlow, the paradigm of a software defined network has expanded beyond the protocol to a full SDN architecture. SDN now forms only one section of the concept of NFV, which seeks to virtualize all parts of enterprise communications technology [18].

2.3 Mininet

Mininet is a popular open-source tool for generating virtual networks whose strength derives from the minimal amount of resources required to run it [23]. Mininet allows for the building of customized SDN topologies through the inclusion of customizable controllers, switches, and hosts. All behavior within the virtualized network is alterable via Python scripts, and allows for changes in bandwidth, latency, and reliability of devices on the network. The program includes support for multiple versions of the OpenFlow protocol. It also supports various types of controllers (NOX [14], OpenDaylight [16]) and switches (pyswitch, Open vSwitch). The program also contains

built in apps such as MiniEdit which can build networks using a Graphical User Interface (GUI), as well as suites for performing ping/traffic tests of built networks.

2.4 Controller Design

As SDN use has expanded, multiple varieties of controllers have been created with different ideas on how the SDN paradigm should be implemented. These range from the simple, custom and centralized, to the robust, open-source and distributed. In this section some of the more popular controllers are identified and described. A more comprehensive list of available controllers can be found at [11].

2.4.1 Ethane.

Ethane [13] was one of the first platforms designed in a modern-day SDN architecture, even before OpenFlow v1.0 was published. Ethane operates in a simplified central controller manner, with a customized language called *Pol-Eth* used for communication between the data and control planes. It operates on a strong consistency model, meaning the controller must be provided with the most up-to-date state of the network at all times.

2.4.2 NOX.

NOX [14] is another centralized controller platform, and one of the first to make use of the OpenFlow protocol. Like Ethane, it also depends on a strongly consistent model of network view to make network control decisions. While this works for a smaller network with a limited number of switches, this can become a liability in larger networks where the amount of time taken to update network state can induce latency. NOX also comes built in with a set of 'system libraries' which could perform a set of network services similar to those that would exist in the Management plane.

2.4.3 Floodlight.

Floodlight is another example of a centralized controller [15]. It differs from other versions in that it is Java-based, giving it greater extensibility as it can operate cross-platform and be more easily modified through an open-source license. It also allows for Management plane applications, written as Java modules, to work with the controller. Floodlight does suffer some limitations as far as maximum number of maintainable flow entries [11].

2.4.4 Onix.

Onix was one of the original distributed controller models [12]. This gave Onix an edge over centralized systems of the time for improved scalability and reliability of the control network. Onix also included a few extra features not previously considered such as consistency models between controllers and fault tolerance. The only real weakness for Onix at this time is its required commercial license. This make use of the controller more prohibitive for experimental or educational users.

2.4.5 OpenDaylight.

OpenDaylight is a rapidly-distributed SDN controller sponsored by The Linux Project that has seen recent rapid adoption [16]. It is an open-source alternative to commercial controllers such as Onix. It is even more freely available compared to most open-source tools, having been licensed by the Eclipse Public License (EPL) as opposed to the competing open-source software license, GNU General Public License (GPL) [24]. OpenDaylight is also more full-featured than other controllers, as it has the ability to support multiple southbound APIs in addition to OpenFlow simultaneously [11]. If OpenDaylight suffers from a weakness, it is one inherent to most SDN controller platforms: lack of security due to no encryption on transmitted

data. This is an issue which is only just starting to be addressed; the details of these vulnerabilities is outside the scope of this research.

2.5 Controller Placement Algorithms - Single Objective

Initial deployments of SDN controllers used a single controller for an entire network [14, 15]. As network size increased, so did the amount of overhead for traffic between the controller and its devices; with enough devices, a controller can become a bottleneck as it becomes unable to keep up with the number of flow requests. To alleviate this concern, as well as to improve the scalability of an SDN, distributed controllers were introduced [12, 16]. This improved both of these traits on an SDN, at the cost of increased data plane to control plane traffic. In addition, packet traffic now exists within the control plane, as controllers need to share metadata of traffic that passes between their devices. Keeping a consistent network view is also required to prevent forwarding loops or accidentally forwarding traffic to non-existent nodes. The amount of bandwidth available for inter-controller communications can be extremely limited. In some cases this traffic can itself become a bottleneck. To address this issue, [25] proposes the use of a two-layer hierarchal system of controllers; the lower layer of controllers is placed in close proximity to the data plane and is responsible for determining and passing instructions to switches, while an upper layer is responsible for maintaining awareness of the state of the network as a whole. This concept was first introduced through Kandoo [26].

Now that the use of distributed controllers has become more common, a new emergent problem is the decision of where to place controllers on a network in order to maximize performance. This is a difficult problem to answer because the definition of 'performance' can differ depending on the person administering the network. For example, in order to reduce communication latency among multiple distributed

controllers, the easiest solution would be to place these controllers as close to each other in the network topology as possible. However, such a decision would increase the communication latency of switches on the far edges of the network, and would in fact defeat the purpose of using distributed controllers in the first place.

Several research papers have been released since 2012 that deal specifically with this 'Controller Placement' problem. All of these articles begin with the premise that choosing the 'optimum' placement for any given number of controllers is an NP-hard problem. Each article uses a different approach to determining and labeling the characteristics most important to optimizing the network, followed by extending these parameters to an algorithm to solve their specific problem. Some of these articles only choose to solve one metric, while others attempt to balance several.

2.5.1 Single Objective Problem - Reliability.

One of the first articles [27] solves the single objective problem of controller placement with respect to maximizing reliability of the network; this is computed within the article as maximizing the expected percentage of valid control paths when a network failure occurs. The parameters of their problem consisted of the following:

- An undirected graph $G = (V, E)$ with the set of all network nodes V and set of edges E.
- Network components $l = V \bigcup E$, with network individual nodes $v \in V$ and individual edges $e \in E$.
- Set of potential failure scenarios S. For simplification, the paper considers each failure scenario to be independent. To further simplify the algorithm, each scenario is implemented individually and only consists of a single v or e failure.
- p_l, the probability of component failure given as a $[0, 1]$ value.
- Link weights, based on latency or distance.

- V_c, the set of available placement locations for controllers, $V_c \subset V$.
- M, the set of chosen controllers within V, $M \subseteq V$.
- c, the set of controller-to-controller links.
- $k = |M|$ Total number of controllers.
- $P(M)$, the final set of chosen controllers.
- δ, the expected percentage of valid control paths when network failures occur.

For additional simplification, the authors make multiple additional assumptions, such as controllers being connected in a full mesh, and all network switches being connected to controllers in the shortest path (determined by latency).

Their final algorithm is depicted as follows. The expected percentage of valid control paths, δ, is defined as

$$\delta = 1 - \frac{1}{m} \sum_{l \in V \bigcup E} \sum_{i \in V, j \in V_c, V_c \subseteq V} h_{ijl} x_{ij} p_l \tag{2.5.1}$$

$$= 1 - \frac{1}{m} \sum_{i \in V, j \in V_c, V_c \subseteq V} x_{ij} c_{ij} \tag{2.5.2}$$

$$c_{ij} = \sum_{l \in V \bigcup E} h_{ijl} p_l \tag{2.5.3}$$

where h_{ijl} denotes whether a network path from i to j passes through l, and c_{ij} is the failure probability of the control path between i and j. The objective function given this formula is to maximize δ, given several additional restrictions [27].

The authors depict the problem as a generalized form of the minimum k-median problem, as well as a form of the k-Singular Value Decomposition (k-SVD) problem [28]. They propose three different algorithms for solving the controller placement problem:

1. Random

2. Iterative greedy by non-increasing reliability (non-decreasing failure rate), with varied backtracking
3. Brute Force (exhaustive)

While the authors use real network topologies to perform their simulations, they make a number of assumptions to simplify computation:

- Each switch connects to exactly one controller.
- No backup mechanism.
- Single device failure scenarios only.
- Every node can be a controller.

Even given these constraints, the brute force method was forced to be limited to two weeks on a quad-core processor for the larger network topologies (104 nodes/302 links and 183 nodes/744 links). The random placement method performed the worst overall. The greedy method managed to come within 5-8% of optimum, with better approximations on smaller networks.

2.5.2 Single Objective - Propagation Delay.

Heller et al. seeks to solve the same problem with a focus on latency as the measure of capability [4]. Their reasoning for this is based on the assumption that the network in question is some type of Wide Area Network (WAN). The authors acknowledge that other metrics such as load balancing or reliability may take priority for data centers or enterprise networks.

This article formulates the problem as follows: consider a network graph $G(V, E)$. Each edge has an edge weight $d(v, s)$ as the shortest path from $v \in V$ to $s \in V$, number of nodes $n = |V|$.

Given these parameters, the authors cite multiple objective functions. Average Latency is defined as

$$L_{avg}(S') = \frac{1}{n} \sum_{v \in V} \min_{(s \in S')} d(v, s) \tag{2.5.4}$$

and is considered a minimization of the k-median problem, where S' is the placement of a controller out of all placement options S.

The counterpart to Average Latency, Worst-Case Latency, is defined as

$$L_{wc}(S') = \max_{(v \in V)} \min_{(s \in S')} d(v, s) \tag{2.5.5}$$

and is the formulation of the minimum k-center problem.

Both equations are designed with the placement of a single controller in mind. A third algorithm option in the paper is to maximize the number of nodes in the network that are within a certain time bound. In this algorithm a number k is introduced as well as a set of sets $S = S_1, S_2, \ldots, S_m$, where $S_i \subseteq v_1, v_2, \ldots, v_n$, each subset representing the nodes which are brought below a given latency bound by the placement of a controller at node S_i. The objective is to choose sets $S' \subseteq S, |S'| = k$ which maximize the number of nodes V below the latency bound. In this sense this problem is formulaic of maximum cover, another NP-hard problem.

To test their results, they used a number of network topologies from the Internet Topology Zoo [10]. Surprisingly most latency concerns could be alleviated with a single controller; however this does not take into account the reliability of the network.

2.5.3 Single Objective - Resilience.

Guo [29] wrote a variation on Hu's paper [27] by introducing the concept of an interdependence graph. Where Hu kept all potential network interruptions separate, Guo created a graphical network demonstrating device dependencies between two different layers of the network: the Switch-Switch (SS) network and the Controller-

Switch (CS) network. When a node a points to a node b on the graph, it is said that a depends on b. Subsequently, if node b fails, then node a also fails. Consider the graph in Figure 4; node 1-SS depends on node 1-CS and node 2-CS on node 2-SS. Nodes 3-SS and 3-CS point to each other, denoting a mutual dependence.

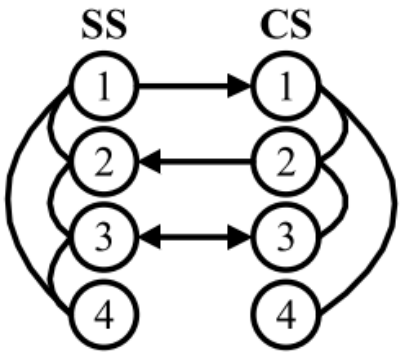

Figure 4. Example SS-CS Dependency Graph [29]

In the case of multi-staged dependencies, it becomes possible for multiple devices to fail in sequence should a device at the head of the dependency list fail. This is known as a cascading failure. If a link does fail, the eventual state of the network after all dependent devices also fail is called the steady-state of the network.

Guo's algorithm is formulated as follows: Consider again the graph $G = (V, E)$, a number of controllers k, a limited set of options for placement of controllers $L \subseteq V$, a directed controller-switch communication link network ψ, and a probability distribution of link/node failure Δ. All of these are used to create an interdependence graph H. We seek to select k nodes in L which maximizes the objective function ϕ

$$\phi = 1 - \frac{1}{|H|} \cdot \sum_{i \in H} \Delta^i \cdot \left[\alpha \sum_{j=1}^{k} f_j^i + (1 - \alpha)\mu^i \right] \tag{2.5.6}$$

where α is a trade-off between sub-network and network resilience priority, Δ^i is the probability of failure of node i, f_j^i is the fraction of nodes assigned to controller j

but not in the mutual cluster of j once steady-state is reached after failure of node i, and μ^i is the fraction of nodes in the largest mutually connected cluster at the steady-state after failure of i.

2.5.4 Multiple Objective - Scalability and Load Balancing.

Jiménez et al. introduce a newly defined algorithm for optimum controller placement, which they call 'k-Critical' [30]. This is a new algorithm with the two-part goal of minimizing the number of controllers needed for the network while still creating a robust and load-balanced network topology. The paper defines the algorithm, then compares it to the previously known NP-Complete problems of k-Center and k-Median [30] to compare performance.

The k-Critical algorithm functions as a two part equation. First, the algorithm determines controller placement via a function that keeps the network layout as flat as possible while keeping controller-switch latency below a pre-determined value, $\mathcal{D}_{req}$. Node suitability for controller placement is determined through the function θ, which is defined as

$$\theta = \gamma \times \frac{\mathcal{D}_g}{N - h_i} + (1 - \gamma) \times \frac{\mathcal{D}_{max}(v, \mathcal{C}_k)}{\mathcal{D}_{req}} \tag{2.5.7}$$

where $\gamma = \dfrac{\mathcal{L}_h}{\mathcal{L}_{max}}$, $\mathcal{L}_h$ = maximum path length for controller node considered, $\mathcal{L}_{max}$ = maximum path length for all controller potential nodes, h_i = number of other nodes the considered node has covered at i hops, D_g is the degree of the considered node, and $\mathcal{D}_{max}(v, \mathcal{C}_k)$ is the maximum delay from the considered node to any other node that it covers. The pseudocode which performs this calculation is shown in Algorithm 1.

The first part of the equation ensures that all controller-switch communication is below the $\mathcal{D}_{req}$ delay. To ensure controller-controller communication is also below this

Algorithm 1 Jiménez Controller Placement Pseudocode Part 1: Controller Selection Algorithm k-Critical Node [30]

Require: $(\mathcal{N} \times \mathcal{N})$ SP Delay Matrix. $\mathcal{D}_{req} \leftarrow$ Req. Delay

$C \leftarrow$ Set of candidate nodes that satisfy $\mathbb{D}_{req}$
if $C \neq \emptyset$ **then**
  for each node $n \in C$ to become a controller **do**
    Evaluate θ
  end for
  Select the node $n \in C$ with the highest value in θ; $C = n$
else
  while there are nodes not belonging to the cluster **do**
    $C_n \leftarrow$ Find the farthest node to the Cluster
    $C_k \leftarrow$ Find the set of candidate nodes to manage C_n
    for each candidate node $n \in C_k$ **do**
      Evaluate θ
    end for
    Select the node n with the highest value in θ
    Cluster$_k \leftarrow$ Find from $(\mathcal{N} \times \mathcal{N})$ the nodes v that satisfy $d(v, n) \leq \mathcal{D}_{req}$, where $v \notin$ Cluster
    Cluster $\leftarrow$ Cluster $\bigcup$ Cluster$_k$, $C = C \bigcup n$, Cluster$_k \leftarrow \emptyset$, $C_n \leftarrow \emptyset$
  end while
end if

delay, a second equation is needed. To do this, the controller-controller delay is quantified as $\mathcal{D}_{cont}$ and seeks to guarantee that $\mathcal{D}_{req} + \mathcal{D}_{cont} \leq \mathcal{D}_{max}$ by adding additional controller nodes if required. The appropriate pseudocode is given in Algorithm 2.

Algorithm 2 Jiménez Controller Placement Pseudocode Part 2: Controller Placement [30]

Require: $(\mathcal{N} \times \mathcal{N})$ SP Delay Matrix. $\mathcal{D}_{cont} \leftarrow$ Delay required among controllers. $\mathcal{D}_{max} \leftarrow$ Maximum delay in network. $C \leftarrow$ controllers previously selected. $k \leftarrow$ Number of controllers in C.

1: **if** $\mathcal{D}_{cont} > \frac{\mathcal{D}_{max}}{k+1}$ **then**
2: $\quad \mathcal{D}_{cont} >= \frac{\mathcal{D}_{max}}{k+1}$
3: **end if**
4: **while** $\mathcal{D}(C_i, C_j) \geq \mathcal{D}_{cont}$ **do**
5: $\quad C_n \leftarrow$ Find the farthest node in the Controller cluster
6: $\quad C_k \leftarrow$ Find the candidate nodes to manage C_n where $C_k \notin C$ and minimize the delay to C
7: $\quad$ **for** each candidate node $n \in C_k$ **do**
8: $\quad\quad$ Evaluate θ
9: $\quad$ **end for**
10: $\quad$ Select the node n with the highest value in θ
11: $\quad C_{up} \leftarrow C \bigcup C_k$, $C_n \leftarrow \theta$
12: **end while**

Jiménez et al. test the performance of the k-Critical algorithm through randomly generated graphs of categorically different connectivity through the use of an open-source program called Gephi [31].

2.5.5 Multiple Objective - Latency, Data Rate, and Processing.

Auroux et al. take a unique approach to solving the controller problem as it does so with a unique perspective on the hardware being used [32]. This paper deals specifically with deciding controller placement while minimizing energy consumption (processing capability of devices) in addition to bandwidth (data rate) and traditional

latency. It considers these things because the network is designed with wireless devices in mind, which typically have lower data capabilities than traditional network hardware. This approach also uses the two-layer controller hierarchy introduced by Yeaganeh [25]. The authors label their lower level controller a CROWD local controller (CLC), responsible for faster flow generation. The upper level controller is called a CROWD regional controller (CRC) responsible for overall network view. These identifiers are in the context of a previous networking project described in [32].

Consider a graph $G = (V, E)$, where every $e \in E$ is unidirectional and has a latency $l_{cap}(u, v)$ seconds and a data rate cap of $b_{cap}(u, v)$ bits. Also consider a controller candidate list $C \subseteq V$ with its own data rate cap $b_{max}(c)$ and processing limit $p_{own}(c)$. Moreover, this paper introduces quantification of the set of flows F, where each flow $x \in F$ induces its own latency $l_{flow}(x)$ and data rate consumption $b_{flow}(x)$.

2.5.6 Single Objective - Monetary Cost.

Sallahi introduced additional objectives restricting network topology based on the cost of hardware an administrator may have available for running a network [33].

New parameters included are a set of *types* of controllers $c \in C$ and links $l \in L$ that may be used, which possesses its own features such as:

- Number of ports: a^c
- Processable packets/second: μ^c
- Controller cost: κ^c
- Number of controller of specific type: φ^c
- Link bandwidth (in Mbps): ω^l
- Cost of link ($/meter): ϕ^l

The goal of [33] is to minimize the overall cost of placing controllers in available locations, linking all switches to controllers, and linking controllers to each other.

Note, however, that this paper does not consider the cost of how switches are linked to each other. Ideally, this algorithm should be modified to include such a concern. This formula is best used when planning to either majorly extend an existing SDN or establish a new one.

2.5.7 Single Objective - Latency w/ Performance Gain.

Tuncer's paper is even more unique from previous works [34]. Similar to [32], it uses the two layer hierarchal controller system presented in [25]. On top of this, it differs from the previously described definition of an SDN architecture by placing controllers and management applications on equal footing in each of the upper two layers, as opposed to having applications sit on top of controllers [11].

In [34], controllers and managers (applications) are considered to be independent entities. These devices sit in a two tier hierarchy similar to the previous examples. At the local layer a controller and a manager are labelled a Local Controller (LC) and Local Managers (LM), respectively. At the higher regional layer, these same devices are considered a Local Controller Orchestrator (LCO) and Local Manager Orchestrator (LMO).

The algorithm presented for solution in this paper is relatively simple compared to previous solutions, as it is designed with the mindset of a static network as opposed to dynamic networks. In addition, their algorithm is greedy, as opposed to an optimal algorithm. The pseudocode is presented in Algorithm 3.

In the evaluation section, the paper denotes how the initial placement has a significant effect on the total number of controllers selected. Characterizing the controllers can help with comparing selection trends.

Algorithm 3 Tuncer Greedy Controller Placement Pseudocode [34]

Require: Link statistics; Splitting ratios

```
1: while adjustments are possible do
2:   Determine the link l_rem and L_flows list of flows traversing l_rem
3:   if L_flows is empty then
4:     End algorithm
5:   else
6:     f ← first flow in L_flows
7:     while canContinue and nbTested ≤ size list L_flows do
8:       Adjust ratios of f
9:       if valid configuration then
10:        canContinue = false
11:      else
12:        f ← L_flows.next()
13:        Increment nbTested
14:      end if
15:    end while
16:    Update link statistics
17:  end if
18: end while
19: return Updated splitting ratios
```

Consider the equation

$$\forall i \in \mathcal{N}, \Delta(i) = \frac{\sum_{j \in \mathcal{N} \setminus \{i\}} d_{i,j}}{|\mathcal{N}|^2} \tag{2.5.8}$$

where $\Delta(i)$ is the average distance of node i in relation to all of the other nodes j, $\mathcal{N}$ is the set of all nodes, and $d_{i,j}$ is the distance between nodes i and j. They determine that an off-center initial controller placement can be more beneficial for smaller networks, while center placement is better for larger networks.

2.5.8 Multiple Objective - Pareto-based Optimal COntroller Placement (POCO).

2.5.8.1 POCO - Enumerative.

Hock et al. has developed a comprehensive testbed for solving the CPP given a larger number of objectives [8, 7] . These papers outright acknowledge the issues that arise when attempting to create a controller placement solution when using metrics that are in conflict.

2.5.8.1.1 The Pareto Front. When given a number of objectives that may be in conflict, it becomes necessary to decide on a solution that makes compromises between the metrics one seeks to optimize. Even so, given a series of solutions, one can eventually reach a set of solutions where it is impossible to improve one metric without degrading another. This set of solutions becomes what is known as the "pareto front." Depending on the problem being solved this front can either be a set of discrete values or a continuous vector. Most fronts are depicted as a trade-off between two or three objectives, allowing the trade-offs to be displayed in a 2D or 3D graph. In reality however, there is no limit to the number of metrics (typically

written as "objective functions") that can be used as limitations to performance.

The concept of a pareto front is now described in more detail based on Van Veldhuizen's paper [35]. First, there is the concept of "pareto dominance". Consider a vector of points $\boldsymbol{u} = (u_1, \ldots, u_p)$ and another vector $\boldsymbol{v} = (v_1, \ldots, v_p)$. $\boldsymbol{u}$ dominates $\boldsymbol{v}$ if and only if $\boldsymbol{u}$ is partially less than $\boldsymbol{v}$, i.e., $\forall i \in \{1, \ldots, p\}, u_i \leq v_i \wedge \exists i \in \{1, \ldots, p\} : u_i < v_i$. Second, there is the additional concept of "pareto optimality", defined as having a solution x_u within set of solutions $\mathcal{U}$ such that there is no alternative solution $x_v \in \mathcal{U}$ where $v = f(x_v) = (v_1, \ldots, p)$ dominates $u = f(x_u) = (u_1, \ldots, p)$.

As demonstrated in the previous papers, there are many different metrics one can use to measure the overall performance of a computer network. To address this, the authors introduced the POCO framework, which is designed to measure network performance on multiple objective functions. The framework is instantiated as an open source GUI program written in MATLAB. Common terms and definitions for POCO are listed in Table 1.

The POCO framework is designed to evaluate the optimum placement of controllers in a variety of network scenarios; specifically one of the following:

1. Failure free - all nodes of the network operate as normal, including controllers.
2. Up to two node failures - specifically defined as any node *not* selected to be one of the k controllers.
3. Up to k-1 controller failures - which is designed to test the effects of nodes being forced to fall back on secondary controllers.

In addition, the POCO framework includes a scenario which can discern the minimum number of k controllers necessary in a given network to guarantee that all nodes are able to communicate with a controller given any two node failures. While a valuable tool in it's own right, it is not the focus of this thesis.

POCO has the capability to evaluate models of networks for a variety of problem

Table 1. A Listing of Terms Used for POCO Framework Equations

Term	Definition
$G = (\mathcal{V}, \mathcal{E})$	Graph with $\mathcal{V}$ nodes and $\mathcal{E}$ edges
k	Number of controllers
$d_{v,w}$	Distance matrix between all nodes $\{v, w\} \in \mathcal{V}$
$\mathcal{P}$	Set of placement of controllers in a network
p	A single controller placement $\in \mathcal{P}$
$\emptyset$	Failure free scenario
$\mathcal{C}$	Set of all possible $k-1$ controller failures
$\mathcal{X}$	Set of all possible 1 to 2 node failures
$\mathcal{N}$	The given node failure scenario for certain metrics
s	A given node failure scenario $\in \mathcal{S}$
$e^s_{v,p}$	A matrix with 1 if nodes v, w cannot reach each other in failure scenario s, 0 otherwise
π	A single metric, further specified with superscripts

instances. The program contains some standard problem profiles to load for a given network topology:

- Calculate best k=1-to-5 controller placements given no risk of controller failure.
- Calculate best k=3-to-5 controller placements given risk of up to two node failures. There is also the option of finding a minimum 'resilient' k value, where resilient is defined as no risk of a node becoming controller-less for any 2 node failures.
- Calculate best k=1-to-5 controller placements given risk of up to $k-1$ controller failures.

In addition, each of these profiles can be altered at the user's discretion by directly selecting, by id #, which nodes are controllers, which controllers fail, and which nodes fail.

Figure 5 provides an example POCO Framework layout demonstrating optimum placements for 5 controllers with no node/controller failures. Controllers are nodes

with double rings. The different color swatches for nodes show which node is connected to which controller, highlighting the controller imbalance metric. The graph on the bottom half of the figure shows the Pareto optimal placements of controllers based on two objectives; each grey dot can be clicked to show the placement of controllers on the graph that gives the indicated results. Note the time at the bottom-right of the POCO window showing the time taken to generate this solution (1111.65 seconds)

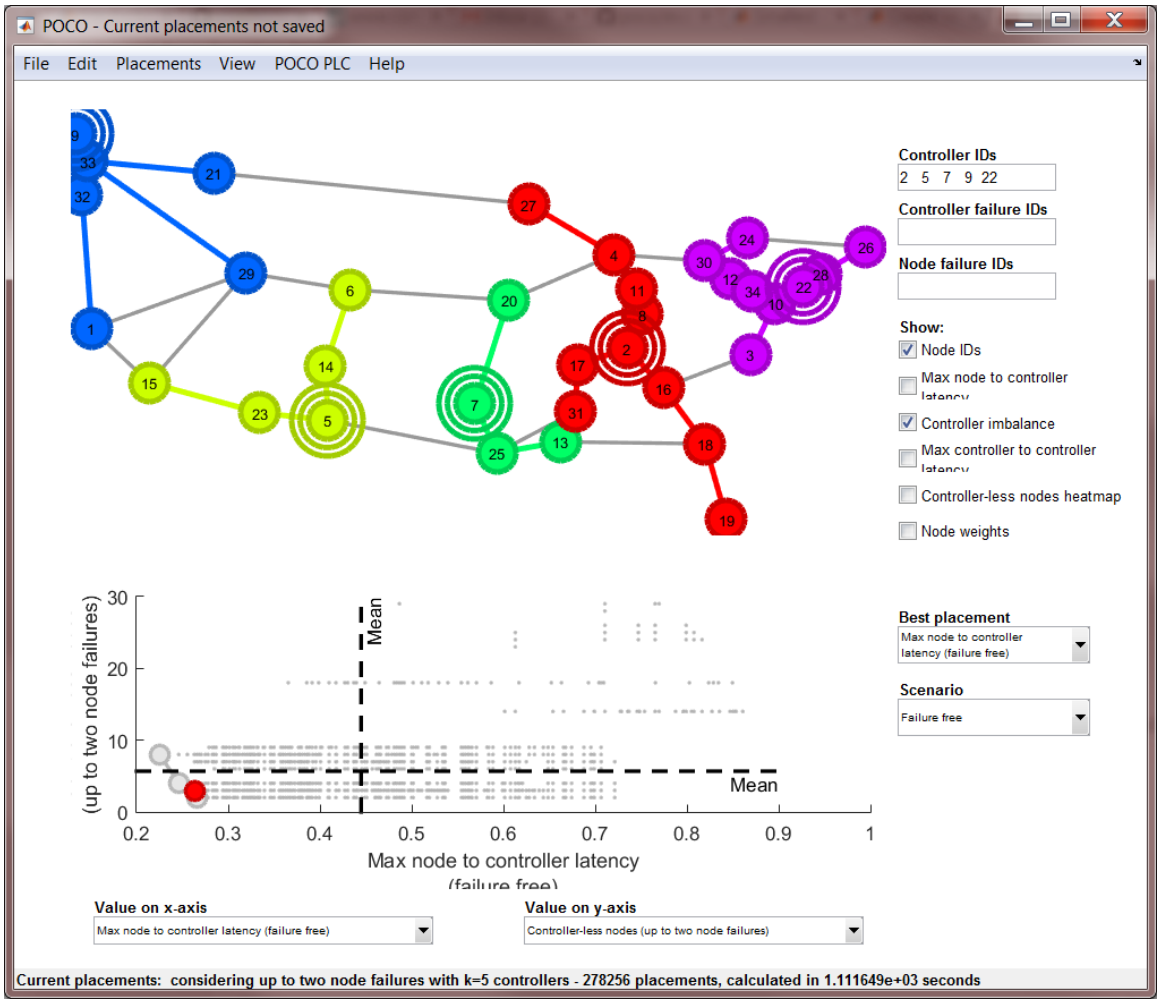

Figure 5. An Example POCO Framework Layout Demonstrating Optimum Placements for Five Controllers with No Node/Controller Failures [7]

POCO framework identifies multiple metrics for measuring network performance; all metrics are minimization objective functions:

1. Maximum node-to-controller latencies: $\pi^{\text{max latency}}(\mathcal{P}) = \max_{(v \in \mathcal{V})} \min_{(p \in \mathcal{P})} d_{v,p}$
   (Note: $k-1$ controller failure scenario returns the worst result out of all controller failure combinations.)
2. Maximum number of controller-less nodes: $\pi_{\mathcal{N}}^{\text{controller-less}}(\mathcal{P}) = \max_{s \in \mathcal{S}} \sum_{v \in V} \min_{p \in \mathcal{P}} e_{v,p}^{s}$
   (Note: s is any given failure scenario $s \in \mathcal{N}$))
3. Controller load imbalance (the difference between the controller with the most assigned nodes and that with the least): $\pi^{imbalance}(\mathcal{P}) = \max_{p \in \mathcal{P}} n_p - \min_{p \in \mathcal{P}} n^p$

The original POCO framework is an enumerative solution generator. Given a network of size N nodes in a graph $G = (V, E)$ network and k controllers, POCO framework evaluates the performance metrics of every possible combination of choosing k nodes to be controllers. Because of this, original POCO is guaranteed to find the optimum pareto front for any given network.

In addition to a program which can determine the optimum placement of controllers in a static network environment [8], Hock et al. also wrote [36] to extend the functionality of POCO to dynamic environments. This paper details the development of POCO-PLC, a version of POCO which contains extra features built to tie directly into the PlanetLab network [37]. This network is a global overlay consisting of servers from hundreds of universities and research institutions for the express purpose of performing network innovation experiments [38]. The PLC functionality is not the focus of this thesis, but it identifies the potential need to dynamically set up and/or teardown controllers based on the changing traffic demands of a live network.

2.5.8.2 POCO - Pareto Simulated Annealing (PSA).

One of Hock's students, Lange, researched one of the most recent, as well as one of the most comprehensive, papers to date with respect to the solving the CPP [39]. This work builds directly on the efforts of [8] and extends the program to include the

use of the Pareto Simulated Annealing (PSA) algorithm. Simulated Annealing (SA) is a metaheuristic search algorithm designed to find approximate solutions to problems which may be computationally intractable (such as this research). PSA operates in an identical manner, but for MOPs. Reasons for using such a metaheuristic for the CPP may include *a*) the network being too large or *b*) the network behavior changing too quickly for an enumerative solution. Using PSA addresses one of the major shortcomings of the enumerative POCO algorithm: namely, that the number of solutions with an enumerative algorithm increases exponentially with respect to the size of the network being solved for. In terms of algorithmic complexity this is known as an NP-hard problem. Use of algorithms like PSA allow us to reach a solution in a much shorter timeframe without sacrificing much in terms of optimality [39].

POCO with PSA [39] includes the use of additional metrics beyond those given in the original POCO paper [8], including:

1. Average node-to-controller latency: $\pi^{\text{avg latency}}(\mathcal{P}) = \max_{v \in V} \min_{p \in P} d_{v,p}$
2. Maximum controller-to-controller latency: $\pi^{\text{max controller-latency}}(\mathcal{P}) = \max_{p_1,p_2 \in \mathcal{P}} d_{p_1,p_2}$
3. Average controller-to-controller latency: $\pi^{\text{avg controller-latency}(\mathcal{P})} = \frac{1}{\binom{|\mathcal{P}|}{2}} \sum_{p_1,p_2 \in \mathcal{P}} d_{p_1,p_2}$

A listing of terms and definitions used for PSA are presented in Table 2. The PSA algorithm begins by first generating an initial, feasible, generally random solution (in this case, the placement of k controllers). Next, a set of random weight values, Λ, is assigned to each solution. Next, consider a selection of 'neighboring' solutions; a neighborhood is a selection of possible alternative solutions (i.e., controller placements) which is some subset of the solution space. For example a neighborhood could be considered all controller placements which only differ from the controller solution by 1 node, or 2, etc. The neighborhood is usually decided at the start of the algorithm.

Another unique feature of PSA is the use of a 'temperature level', T to increase

Table 2. Terms Specific to the Pareto Simulated Annealing (PSA) Algorithm

Term	Definition
Λ	Set of weights assigned to set of solutions S
M	Set of Pareto solutions from S
T	Temperature of solution
T_0	Starting temperature
T_ρ	New temperature based on PSA formula
Y	Set of neighboring solutions to solutions within S
P	Probability of accepting a new solution

the variety of solutions that are acceptable. When a new solution is generated in PSA, it may be better or worse than the current solution. If the solution is better, it is universally accepted. If the solution is worse, it may still be accepted and replace the previous solution, based on a probability dependent on a formula $P(S, Y, T, \Lambda)$. The higher the temperature, the greater probability that the worse solution is accepted. This allows the algorithm to move more easily around the solution space and avoid becoming stuck in a 'locally optimum' solution of a given neighborhood in the search space. After a certain number of generations (line 11 in Algorithm 4), the temperature drops to a certain value T_ρ. The rate at which T_ρ decreases is determined *a priori* as parameter inputs at the start of the algorithm along with T_0. As the temperature drops closer to some threshold (in this case $T = 1$), acceptances of worse solutions become less common, causing the set of solutions to settle, hopefully to a set of more globally-optimum solutions.

Consider the pseudocode given in Algorithm 4. The user starts with an original graph $G = (V, E)$, a number of controllers k, a set of controller placements to consider s, a number of iterations at each temperature level m, an initial temperature level T_0, and a factor of temperature reduction per temperature drop ρ. An initial set solution is generated S, followed by a given weight Λ which influences the probability of

accepting a worse solution, then a Pareto Frontier M (a listing of visited placements). At each generation, the list of available neighbors is generated, the Pareto Frontier is updated, weights are updated, and the new solution S accepted depending on probability P. This continues until the temperature threshold is reached, whereupon the Pareto Frontier and final placement is returned. The results of [39] show that when the search space of possible solutions grows above $4 \times 10^{+6}$ solutions, PSA can achieve comparable results to exhaustive search in a fraction of the time. This difference increases by multiples as the size of the search space increases.

Algorithm 4 Pareto Simulated Annealing (PSA) Pseudocode [39]

Require: $G = (V, E), k, s, m, T_0, \rho$

$n = |V|$
$S = generateRandomPlacements(n, s, k)$
$\Lambda = generateRandomWeights(S)$
$M = paretoFrontier(evaluatePlacements(S))$
$T = T_0$
while $T > 1$ **do**
  $Y = drawNeighbors(S, n, \lceil \frac{kT}{2T_0} \rceil)$
  $updateParetoFrontier(M, Y)$
  $\Lambda = updateWeights(S)$
  $S :=$ accept $y \in Y$ with probability $P(S, Y, T, \Lambda)$
  if m iterations were performed at T **then**
    $T = T\rho$
  end if
end while
return M and corresponding placements

2.5.8.3 POCO - k-Medoids.

Another version of POCO recently published by Lange [40] introduces a different heuristic meant to further speed up the solution process when dealing with certain specialized use cases of networks. While the previous POCO articles dealt with up to six competing metrics, this work focuses on only two. The first is Average Node-

Table 3. Terms and Definitions Specific to k-Medoids

Term	Definition
V	Choice of k controllers in network
A	Specific assignment of k controllers in network
p	A specific controller
n_p	Number of controllers assigned to controller p
ρ	Capacity bound: Maximum quantity of nodes assignable to a single controller
N	Set of nodes in the network
F	Set of controllers replicated according to bipartite matching

to-Controller Latency, defined as

$$\pi^{\text{avg latency}}(\mathcal{P}) = \frac{1}{|V|}\sum_{v\in V}\left(\min_{p\in\mathcal{P}} d_{v,p}\right) \tag{2.5.9}$$

There also exists a variation on this metric with an explicit controller assignment A, defined as

$$\pi^{\text{avg latency}}(\mathcal{P}, A) = \frac{1}{|V|}\sum_{v\in V} d_{v,A(v)} \tag{2.5.10}$$

The second metric is Controller Assignment Imbalance, defined as

$$\pi^{\text{imbalance}} = \max_{p\in\mathcal{P}} n_p - \min_{p\in\mathcal{P}} n_p \tag{2.5.11}$$

The linchpin of this approximation algorithm is the use of ρ, which limits the maximum number of nodes that can be assigned to a single controller in a network. Essentially, this value can be altered in an iterative loop to restrict or loosen the value of $\max_{p\in\mathcal{P}} n_p$ from the controller imbalance objective function. This value is used in lines 2-4 of Algorithm 5 to create a perfect bipartite matching between the set of nodes (N) and the set of controllers replicated in quantities set in part by ρ. Line 5 readjusts the

'centers' of controller clusters in an attempt to improve the 'costs' of these controller placements (in other words, minimization of objective functions). Lines 6-11 repeat this process until this shifting of the clusters no longer offers improvement.

Algorithm 5 Capacitated k-Medoids Pseudocode [40]

Require: D, n, k, ρ

1: $C = \text{kMedoids}(D,n,k)$ $(= \{c_1, \ldots, c_k\})$
2: $N = \{1, \ldots, n\}$
3: $F = \left\{ c_1^1, c_1^2, \ldots, c^{\lceil \frac{n}{k} \rceil + \rho}, c_2^1, \ldots, c_k^{\frac{n}{k}+\rho} \right\}$
4: $(A,\ costs) = \text{match}(N, F, D)$
5: $(C', costs') = \text{recalculateCenters}(A)$
6: **while** $costs' < costs$ **do**
7: $\quad C = C'$
8: $\quad F = \left\{ c_1^1, \ldots, c^{\lceil \frac{n}{k} \rceil + \rho} \right\}$
9: $\quad (A, costs) = \text{match}(N, F, D)$
10: $\quad (C', costs') = \text{recalculateCenters}(A)$
11: **end while**
12: **return** (C, A)

While Algorithm 5 can provide improvements for a specific limit on the allowed differences between controller assignments, it is Algorithm 6 which provides the creation of a true pareto front based on an iteration of ρ values.

Algorithm 6 Pareto Capacitated k-Medoids Loop [40]

Require: D, n, k, P, n_r

1: $S = \emptyset$
2: **for all** $i \in \{1, \ldots, n_r\}$ **do**
3: $\quad$ **for all** $\rho \in \mathrm{P}$ **do**
4: $\quad\quad S = S \bigcup \text{capactitatedKMedoids}(D, n, k, \rho)$
5: $\quad$ **end for**
6: **end for**
7: $S = \{s \in S | s \in \text{paretoFrontier}(\text{evaluate}(S))\ \}$
8: **return** S

2.5.9 Alternative Platforms.

At its core, the CPP is a specialized case of the *Plant, Warehouse, or Facility Location Problem* [5], which represents any scenario in which one seeks to choose the 'best' location for a certain resource (controllers in this case) out of a finite number of options for a network [4]. Alternative tools exist for solving this type of problem. Dr. Zinner et. al. note that IBM has written a program called the *IBM ILOG CPLEX*, which can solve this type of problem in general cases. However, it is not designed to handle specialized problems such as controller placement for an SDN.

Mininet is a Python-based program which can create a network of controller, switch and host nodes within a Virtual Machine (VM) [23]. This network can then be put through a suite of communications tests to determine device performance. In addition, a variety of controller software can be used for comparative purposes. Unfortunately, the program does not at this time have a way to perform network evaluation based on selecting certain nodes to be the controllers. It is this author's opinion that such a platform could be immensely powerful in providing knowledge of how different networks would perform in the future. The work required for this addressed in Chapter VI of this thesis.

GENI [41] is a virtualized network similar to PlanetLab which houses servers in multiple education centers across the United States for the purposes of distributed networking experiments. The authors cite the use of their network for the experimentation of SDN set deployments, as well as many network design choices based on PlanetLab.

2.5.10 Summary.

Out of the heuristic solutions provided to date, PSA provides the most comprehensive algorithm in terms of accounting for as many objectives as possible. Indeed, it is

the only version of the controller placement problem that qualifies as a Many Objective Evolutionary Algorithm (MaOEA) [42], and possesses strengths such as running relatively quickly. However, as it is the only known MaOEA for the controller placement problem, it remains to be seen if PSA is the most optimal metaheuristic. In the next chapter an alternative to PSA, POCO-MOEA, is formally defined.

III. Methodology

3.1 Overview

This chapter presents the foundation upon which POCO-MOEA is built, as well as providing a formal function definition of the algorithm. First, Section 3.2 gives an explanation of how a general MOEA is defined, and provides an example structure through the NSGA-II algorithm. Next, POCO-MOEA is formally defined through the building of the Problem Domain, Algorithm Domain, and pseudocode explaining the order of operations. Explanations of the fundamental MOEA operators used in POCO-MOEA are also explored. Finally, the algorithm's computational complexity is analyzed.

3.2 MOEA Explained

In this section the algorithm elements used in POCO-MOEA are explained. As the foundation of POCO-MOEA is based on NSGA-II, it will use similar components for performing its operations. These are:

- Population - Contains a limited number of possible solutions for controller placement for a given scenario.
- Solution/Individual - A single selection of k controller placements within a network from the Population.
- Chromosome - A single piece of the solution; in this case, a single controller out of the set of controllers within an individual.
- Crossover - A probability for crossover between two different solutions within the population.
- Mutation - A probability for exchanging a controller within the current solution to a different node within the network.

- Fitness - The calculated Rank for a given solution compared to the other solutions.
- Crowding Distance - Influences the rank given to each solution based on its proximity to other solutions. Solutions near the edge of P_{front} and those isolated from other solutions receive better ranks.
- Generation - The population of solutions at a given point in time.
- Stopping Condition - condition upon which the algorithm ceases and the results are provided to the user. Conditions can include a certain number of generations or when the population reaches a certain level of stability.

An example of how an MOEA is structured and executed is now provided with NSGA-II. This MOEA finds the Pareto front P_{known} of non-dominated solutions among all objective functions within a limited number of generations. The pseudocode in Algorithm 7 shows the order of operations for NSGA-II; it has been rewritten for clarity based on the pseudocode given in Deb's original article [9].

Three of the most important components for any MOEA are the crossover, mutation and selection operators. The crossover operator simulates the inherent DNA sharing between two parents and acts as a way of mixing information between two solutions in the hopes of producing a better one. Mutation simulates the changes an individual undergoes during its growth that may cause it to differ even further from its parents. Finally, selection simulates the idea of limited resources (in this case the size of the population) forcing only the best solutions to be chosen from generation to generation.

For the purposes of the experiment, simple versions of each of these operators are used. For crossover, the 'one-point crossover' method is implemented, where a certain point in the array of selected nodes for controllers of one parent is chosen, and every node after that point is swapped with the nodes of another randomly selected parent. For the mutation operator the '2-opt' method is utilized, where a random node in the

Algorithm 7 NSGA-II Algorithm for Controller Placement

1: **Procedure:** Produce a set of solutions $\mathcal{N}$ to solve the problem $f_k(\mathbf{x})$ after g generations
2: Initialize population $\mathbb{P}$.
3: Generate $\mathcal{N}$ random solutions for Population $\mathbb{P}$
4: Evaluate objective values
5: Assign rank (level) based on Pareto Dominance
6: Sort population $\mathbb{P}$ based on rank
7: Generate Child Population
8: Binary Tournament Selection
9: Recombination and Mutation
10: **for** $i = 1$ to g **do**
11: **for** each Parent and Child in Population $\mathbb{P}$ **do**
12: Assign rank (level) based on Pareto Dominance
13: Generate sets of non-dominated vectors along PF_{known}
14: Loop (inside) by adding solutions to next generation starting from the *first* front until $\mathcal{N}$ individuals found determine crowding distance between points on each front
15: **end for**
16: Select points (elitist) on the lower front (with lower rank) and are outside a crowding distance
17: Create next generation
18: Binary Tournament Selection
19: Recombination and Mutation
20: **end for**

array of k controllers is exchanged with a node that is not part of the chosen controller array. For the Tournament Selection operator, a general pareto front selection method is used combined with a crowding distance function to promote diversity.

3.3 POCO-MOEA: Formal Algorithm Design

The POCO-MOEA algorithm used for this project is modeled after the original NSGA-II algorithm [9]. The base version of NSGA-II cannot be implemented directly

for this problem due to the data types used for the CPP, i.e., a limited graph environment with integers used for node selection. Specifically, NSGA-II is designed to deal with either a binary solution space or a real (continuous) solution space. The controller placement problem as implemented here deals with a discrete, non-infinite number of choices. Therefore, some of the operations are different from those defined in the NSGA-II algorithm.

The formal algorithm definition progresses over a series of steps from a rough outline to identifying specific components of each part of the algorithm. First, the Problem Domain is explained, which defines the objective to be solved as well as the environment within which the problem resides. Then, the Algorithm Domain is described, which details the design strategy behind the algorithm, followed by the data and algorithm components used to build the solution set and provide an answer to the given problem. Lastly, the algorithm pseudocode provides a summarized series of steps POCO-MOEA works through to accomplish the task at hand.

3.3.1 Problem Domain Design.

First the Problem Domain is defined, otherwise known as the 'solution space' of the algorithm. This is the scope of the problem as described with units and data structures, and is the most basic, fundamental portion of building the algorithm. The purpose of defining the Problem Domain is to provide a bound for the type of information that can be input to the algorithm, as well as limit what kind of solution can be expected from the output. Table 4 identifies the input and output domains of the solution, as well as the respective components.

Within the scope of the Problem Domain, there are the following operational constraints:

1. *Feasibility:* All sets of solutions during the algorithm ($\mathcal{P}$) must consist of nodes

Table 4. Terms and Definitions of the MOEA Problem Domains

Domains	Definition
Input: $\mathcal{D}_i(G(V,E))$	Input domain where G is an undirected network graph with V nodes and E edges
Output: $\mathcal{D}_o(\mathcal{P}_o)$	Output domain where $\mathcal{P}_o$ is the set of final placement solutions

which are actually present in V.

2. *Solution:* The final algorithm output must contain some solution $\mathcal{P}_o$. Solution must consist of integer values identifying the chosen nodes of graph G.
3. *Objective:* The objective of POCO-MOEA is to calculate and output a set of pareto-optimal solutions $\mathcal{P}_o$ to solve the CPP.

3.3.2 Algorithm Domain Design and Specification.

This section exhibits how POCO-MOEA is designed and built to accomplish the objective declared in the Problem Domain section, while staying within the stated feasibility limitations. This section also explains how data is defined, organized and manipulated over the course of POCO-MOEA.

A design 'strategy' for how to produce a solution must first be defined. This design outline provides the framework necessary to articulate the more specific components of POCO-MOEA. The outline components are given in Table 5.

After the basic design phase, the algorithm must be further articulated in how the data is manipulated to produce the final set of solutions. First, the data used within the problem needs to be identified. These are given in Table 6. Graph components G, V, and E all make up part of the input domain given in the Problem Domain. Each set of solutions $\mathcal{P}$ is an $N \times k$ matrix where each row is a single solution of controller placements p within the graph of nodes V. Because each of these solutions are generated after algorithm start, they are not technically part of the input domain,

Table 5. Basic Design Components of the POCO-MOEA Algorithm Domain

Component	Purpose
Set of Solutions	A set of introductory solutions to be shaped
Next State Generator	Operation(s) that generate some new set of solutions
Feasibility Function	Ensures that solutions created by Next State Generator are valid
Solution Function	Provide the best overall set of solutions from those given by Set of Solutions and those developed by Next State Generator
Objective Function	Provide an output of the best generated solutions to the controller placement problem

even though the output domain contains $\mathcal{P}_o$ as its component. N, M, p_c and p_m are all data values used for the algorithm components of the Algorithm Domain specification. k, N, M, p_c and p_m are all chosen by the user at algorithm start.

Following the definition of basic data components, the actual algorithm components within POCO-MOEA must be established. These components comprise the actual data manipulation of POCO-MOEA and fulfill the Next State Generator, Feasibility Function, Solution Function and Objective Function portions of the algorithm design strategy. The full listing of each component is given in Table 7. The following subsections describe how each algorithm component functions as well their purpose.

3.3.2.1 Crossover Operator.

The crossover operator is a major algorithm component of POCO-MOEA; As such, it receives the unmodified set of solutions at the start of each generation (referred to as the 'parent' generation). The crossover operator starts by iterating through each member p within the set of solutions $\mathcal{P}$. On each loop, the crossover probability p_c makes the determination on whether a given member is involved in a crossover. If the crossover occurs, a second parent is chosen at random from every other member of the set of solutions.

Table 6. Terms and Definitions for Data Components of the POCO-MOEA Algorithm Domain

Data	Definition
$G = (V, E)$	An undirected network graph
V	The set of all nodes that make up the network graph; all nodes are identified as points on a 2-dimensional x, y plane
E	The set of all edges connecting pairs of nodes in the graph
$p \in V$	A single solution (i.e. placement) of controllers within V
$\mathcal{P}$	Any set of solutions; also called a Population
$\mathcal{P}_i \in \mathcal{P}$	Set of randomized solutions created at algorithm start
$\mathcal{P}_p \in \mathcal{P}$	Set of intermediate placement solutions created mid-algorithm
$\mathcal{P}_o \in \mathcal{P}$	Final output set of placement solutions
$k = \|p\|$	Number of controllers in the graph
$N = \|\mathcal{P}\|$	The size of the solution sets that are considered at each generation, and as the final output
M	The number of iterations through each algorithm component that are run to create the output population; also known as generations
p_c	Crossover Probability: the fractional probability that two different parents use the crossover operator
p_m	Mutation Probability: the fractional probability that each node within each solution mutates

The operator runs a variation of the single-point crossover, as illustrated in Figure 6. A single point is chosen at random within the solution for both selected members as the crossover point. From this point to the end of the array, the nodes for each parent are swapped, one node at a time.

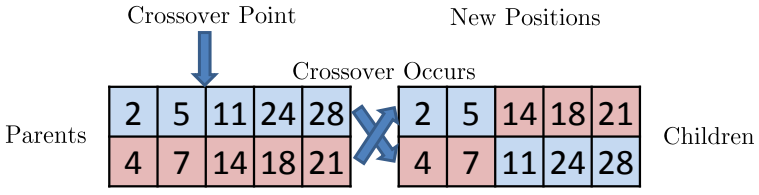

Figure 6. Example of Single Point Crossover for Multiple Network Nodes

Table 7. Terms and Definitions for Algorithm Components of the POCO-MOEA Algorithm Domain

Term	Data Used	Definition
Crossover	$\mathcal{P}_i, \mathcal{P}_p, p_c$	Performs a crossover operation between two members of one solution set (parents) to produce two new members for a new solution set (children)
Crossover Feasibility	$\mathcal{P}_i, \mathcal{P}_p$	Verifies that a solution set contains only non-duplicating, feasible selections for all $p \in \mathcal{P}_i, \mathcal{P}_p$
Mutation	$\mathcal{P}_i, \mathcal{P}_p, p_m$	Mutates individually selected nodes of each solution $p \in \mathcal{P}_i, \mathcal{P}_p$ of the children to some other node within V
Mutation Feasibility	$\mathcal{P}_i, \mathcal{P}_p$	Verifies that the individual nodes of any solution p are feasible (within V) and non-duplicating
Tournament Selection	$\mathcal{P}_i, \mathcal{P}_p$	Selects the best solutions out of the parents and children based on pareto dominance for the next generation
Crowding Distance	$\mathcal{P}_i, \mathcal{P}_p$	Sorts the last pareto front for Tournament Selection to encourage diversity of the next generations' set of solutions
Solution	$\mathcal{P}_p$	Output of intermediate solution set at each generation of the MOEA
Objective	$\mathcal{P}_o$	Final output set of controller placements after all generations M

There are exceptions to this procedure, however. Because there is a finite number of possible nodes to choose as controllers, there is always the risk that a crossover operation causes one or both children to contain duplicate nodes. There is a way to deal with this however, referred to as a subset permutation operator. This is where the Crossover Feasibility portion of the algorithm components comes into play. If a crossover at any point in the aforementioned iterative loop would cause a node to be duplicated in either child, the crossover does not take place for that specific point.

At the end of the crossover operator, every solution $p \in \mathcal{P}$, regardless of whether they were involved in a crossover, becomes part of a set of 'children' solutions. This new set of solutions becomes more important as the algorithm progresses.

3.3.2.2 Mutation Operator.

The mutation operator is the simplest of the three EA operators. The program runs through a single nested loop for each node in each member of the children set of solutions and tests for mutation based on the Mutation Probability p_m. As shown in Figure 7, if the condition is met, the node is changed to a random viable node that does not match any of the other nodes for the chosen p. In order to provide as much distinction between the parent and the child generations, as well as to prevent the loss of good solutions within the parent generation, the mutation operator is only performed on members of the child population.

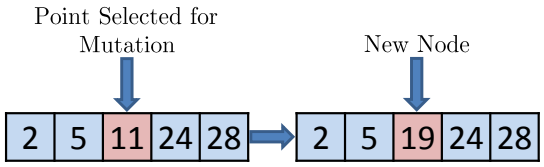

Figure 7. Example of Uniform Mutation for Network Nodes

3.3.2.3 Tournament Selection.

Another major algorithm component of POCO-MOEA is the Tournament Selection operator. In many ways this is the most important component of the algorithm, as it makes the actual determination of which solutions from the parent and children sets are chosen for the parent population of the next loop of the algorithm.

For tournament selection, behavior very similar to that used by NSGA-II is implemented. First, the parent and children sets of solutions are combined to make a single set of solutions, double the size of the initial parent set. Next, the metric values for each objective function of each p within the combined set of solutions is calculated. This is the point in the algorithm where it is actually determined how 'optimum' each solution p within the combined set of parents and children are. These metric values are based on the same six objective minimization functions referenced by POCO and POCO-PSA in Sections 2.5.8.1 and 2.5.8.2, respectively. For reference, these minimization functions are:

- Maximum node-to-controller latency
- Average node-to-controller latency
- Maximum controller-to-controller latency
- Average controller-to-controller latency
- Maximum number of controller-less nodes
- Controller load imbalance

With these metric values retrieved, the Tournament Selection operator then runs through the following loop. First, a pareto front of solutions from the combined parent and child populations is determined based on each solutions set of metrics. This is done using a pareto front operator developed by Dr. Cao [43]. This set of solutions is automatically added to the population for the next generation and removed from the combined parent/children set of solutions. If the size of the population of the

next generation has not exceeded the size of the original parent population, a new pareto front is calculated and the process repeats. Otherwise, the crowding distance operator comes into play.

In the original version of POCO-MOEA, no crowding distance operator is included. This quickly becomes problematic in early testing, as the lack of a diversity operator eventually causes the population to converge to a single solution. The new crowding distance operator seeks to solve this problem by sorting the last pareto front created during the tournament selection operator loop (called $front_{last}$ for this discussion).

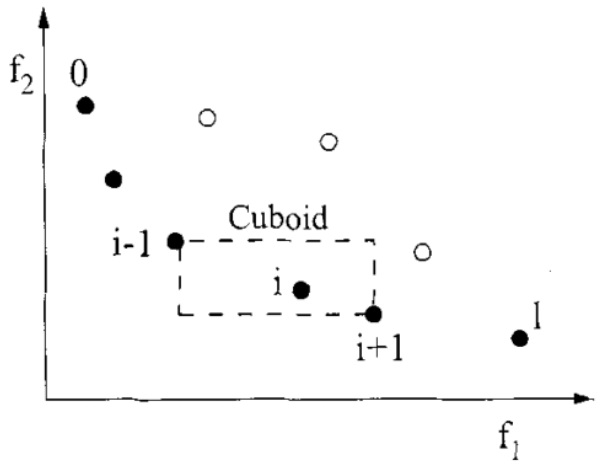

Figure 8. Example of Crowding Distance Calculation [9]

The crowding distance function iterates through each of the six objective functions in the following steps: first, the members of $front_{last}$ are sorted by the current objective function's values in non-decreasing order. Those solutions with the minimum and maximum values for the current objective function are automatically given an 'Infinite' crowding distance. For all other solutions, a crowding distance value is calculated based on the difference of the objective function value between the next highest and next lowest solution in $front_{last}$. A 2-dimensional example of this calculation is shown in Figure 8; the crowding distance for node i is based on the differences

between the f_1 and f_2 values for $i-1$ and $i+1$. This value is summed with a solution's calculated crowding distance value from previous objective functions. This process then repeats for each objective function. A higher crowding distance value represents that a solution is further away from other solutions within $front_{last}$.

3.3.2.3.1 POCO Objective Function Computation. The computation of objective function values and its impact on execution time for both the original POCO exhaustive search and POCO-MOEA are a valuable topic of discussion. In the original POCO and the PSA heuristic, calculation of the objective functions to determine the solution pareto front constitutes a large portion of the code base, and therefore has high impact on overall computation time. Pilot studies find that POCO runs relatively fast when network size is small (approximately 10 seconds for a 35 node network), but can become relatively long if the network is large. For example, performing an evaluation test for the two-node failure scenario with 5 controllers on a 34 node network resulted in an 1,111 second computation time with the original enumerative POCO software on a Core i7 laptop with 16GB of RAM. In another example, a Pareto front placement for 5 controllers in a very large network (approximately 735 nodes) was sought using the same laptop. The system ran out of memory on this attempt; this provides more indication that the time and resources necessary to complete a network evaluation increases exponentially with network size and complexity. When the software was transferred to a server class computer (specifications provided in Section 4.2.3), this same test completed with times typically over 21000 seconds, or nearly 6 hours. Due to the time necessary to complete these trials, a full suite of 30 experiments was not able to be completed in time to provide comparisons to POCO-MOEA. This has been left to Future Work in Chapter VI.

After the crowding distance operator, the pareto front members are sorted by their crowding distances in decreasing order. Solutions with higher crowding distance

values are moved to the front of the array of solutions from the last pareto front. The reason for doing this is based on actions performed at the end of the crowding distance operator. After the sorting of the last pareto front has taken place, the entire set of solutions for the next generation of POCO-MOEA is truncated to become the same size as that of the original size of the parent population. When this occurs, those members of $front_{last}$ with a lower crowding distance are removed from the next generation's population. Those solutions with a higher crowding distance are preserved; solutions with an 'Infinite' crowding distance thus are almost guaranteed to become part of the next generation. In performing this operation, some solutions which are spread further away from each other are preserved for the next generation, encouraging a wider diversity of solutions to draw from for the next iteration of POCO-MOEA operators.

3.3.3 Algorithm Pseudocode.

In this subsection a simplified version of the code is presented and explained to show how characteristics and input parameters are run through procedures to provide the final set of output solutions. The pseudocode is presented in Algorithm 8. This code is a modified version of the NSGA-II code given in Algorithm 7.

3.4 Programming Structure - MATLAB Data Representation

POCO framework is designed to work with a variety of graph file formats, including GraphML (*.graphml), SNDLib topologies (*.xml), or custom topologies within MATLAB (*.topo.mat). GraphML and SNDLib have their own standardized formats, so they are not reviewed here, but the custom MATLAB format is described in more detail.

Algorithm 8 POCO-MOEA Pseudocode

Require: $popSize, k, N, p_c, p_m, distanceMatrix$

Generate starting population $\mathcal{P}_i$
Initialize empty next generation $\mathcal{P}_p$
while Generations run $< N$ **do**
  for Each parent in parent population **do**
    if $rand(1) < p_c$ **then**
      Perform crossover with randomly chosen parent to generate children
    end if
  end for
  for Each node in each member of child population **do**
    if $rand(1) < p_m$ **then**
      Perform mutation operator on selected node
    end if
  end for
  Pull fitness metrics for parents and children
  while New generation size $<$ popSize **do**
    Find current pareto front P_{front} based on parent and children metrics
    if $|\mathcal{P}_p| + |P_{front}| > |\mathcal{P}_i|$ **then**
      Perform crowding distance sorting on P_{front}
    end if
  end while
end while

Networks within .topo.mat files are represented as a series of tables, where N represents the number of nodes and $|M|$ represents number of links. Some of these data points are provided by the input topology file; some are generated during runtime:

- coordinates - n-by-2 matrix of the x,y coordinates of each node in the graph. Also known as 'latlong' for the use of PlanetLab.
- distanceMatrix - n-by-n matrix of the distance between every pair of nodes. Values are normalized based on the diameter of the graph. These distance values also represent the two-way 'latency' between nodes.
- topology - n-by-n matrix identifying links between nodes. An existing link between nodes (i, j) contains the same distance in the corresponding node of the distanceMatrix table. Lack of a link between nodes (x, y) is represented by 'Inf' for infinity.
- maps - a graphical file which can fit behind the graph of a network to provide visual representation.
- nodenames - n-by-1 matrix of string values identifying each node.
- tm - 1-by-n matrix of values which provide weighting data for the nodes of the network. The data for these representations can be arbitrary; in the case of the example provided topology, each value signified the population of the city at each node.
- nk$x$_y - a runtime generated table consisting of every possible combination of k chosen controllers out of n available nodes, by node id #. This can quickly become computationally prohibitive for medium-to-large sized networks and is only used for exhaustive search.

Each of these data structures are manipulated by POCO to generate P_{true} for all objective functions, where P_{true} is the optimum pareto front for a given network.

3.5 Algorithm Complexity Analysis

Building the initial population takes $O(N * k)$ time. The Crossover Operator runs in $O(N)$ time. The Mutation Operator takes $O(N * M)$ time. The Tournament Selection Operator takes $O((N-1) + (N-1) * M)$. Running the entire algorithm takes $O(M)$ generations. Summing all algorithm components together, the entire algorithm runs in $O(N * M)$ time. Actual run time can be highly variable depending on the size of the pareto fronts in each generation, as well as the chosen values for N, M, k, p_c, p_m during problem creation.

3.6 Summary

This chapter covered the components of POCO-MOEA, based on the design of NSGA-II. It also covered the multi-step process behind the development of the algorithm, including basic components, detailed components and pseudocode. The chapter concluded with the algorithm's complexity analysis.

IV. Experimental Design and Methodology

4.1 Goals

In the original version of this experimental design, the goal was to provide a direct comparison between POCO-MOEA and the other heuristics developed for POCO (PSA [39] and kMed [40]). This comparison proved difficult due to the number of variables which can be adjusted for each of the three heuristics. The amount of versatility provided by all three options means that performance can be widely varied by adjusting these parameters. Therefore, this set of experiments has been simplified to only include those of POCO-MOEA and how it compares to exhaustive search.

To allow for a factorial set of experiments within a reasonable time frame, a limited number of parameters are altered for the purposes of this experiment. The experiments seek to answer the following questions:

1. MOEA Efficiency: How do POCO-MOEA input variables affect the computational time of the heuristic?
2. MOEA Performance:
   (a) How much of a time improvement does POCO-MOEA provide over exhaustive search?
   (b) How closely can the pareto front of a POCO-MOEA population approach that of exhaustive search?

By it's very nature, the acceptability of heuristics output depends on the tolerances of the experimental designer. As the output of a heuristic is an approximation of optimal values, it is up to the designer to determine whether such outputs are acceptable in terms of their goals with the system under test (in this case, a network with adjustable controller placements).

Table 8. Set of Full-Factorial Parameters for the POCO-MOEA Algorithm Test

Variable	Chosen Variables
Network Topology Name	Test, Roedunet, Missouri, Latnet
POCO-MOEA Population Size N	200, 800, 8000
POCO-MOEA Generation Count M	1, 10, 50, 100

4.2 Experiment Design

To determine how changing input parameters affects the behavior of the algorithm, a factorial experiment for two of the five variable parameters of POCO-MOEA are run. Each set of variables is run 30 times to test reliability of the algorithm. These variable choices are given in Table 8.

The reasoning for this choice of variables is based on pilot data runs, which are provided in Appendix B. In early test runs of the PSA heuristic, default experimental parameters tended to provide a population size of between 200 and 300 solutions. To provide a similar size of data sets for future comparisons of POCO-MOEA to PSA and other heuristics, a population size of $N = 200$ is selected as a lower bound. In preliminary data tests, the size of the population also seemed to have a significant impact on the various metrics (to be discussed in Section 4.2.4). For this reason progressively larger population sizes have been chosen which are still a small fraction of the exhaustive population size for each network. In regards to the generation count M, a count of 1 is given to show the results of the initial randomly generated population. This provides a baseline to show the metric improvement brought by an increased M value. Previous generation runs also involved M values of 40 and 400. Outputs showed that there was very little difference in some metrics between these two M values (δ_1, δ_2), and slightly improved metrics in others (hyp_{rel}). To test if a smaller generation difference produces the same effects, the values 50 and 100 are chosen as upper bounds. Lastly, the value 10 is chosen as an intermediate value to

show the progression between 1 and 50 generations.

A diagram illustrating the POCO-MOEA System Under Test (SUT)is provided in Figure 9. To reduce the number of necessary experimental runs for a true full-factorial experiment, other variables have been fixed. The Crossover Probability p_c remains fixed at 0.9, the same default value provided by the creators of the NSGA-II algorithm [9]. The value of k for the number of controllers is limited to 5 for all experiments. Because of this, the Mutation Probability p_m is fixed at 0.2, which is equivalent to $1/\ell$ (where ℓ is the number of decision variables [9]). In this case, $\ell = 5$ to represent the number of controllers being placed.

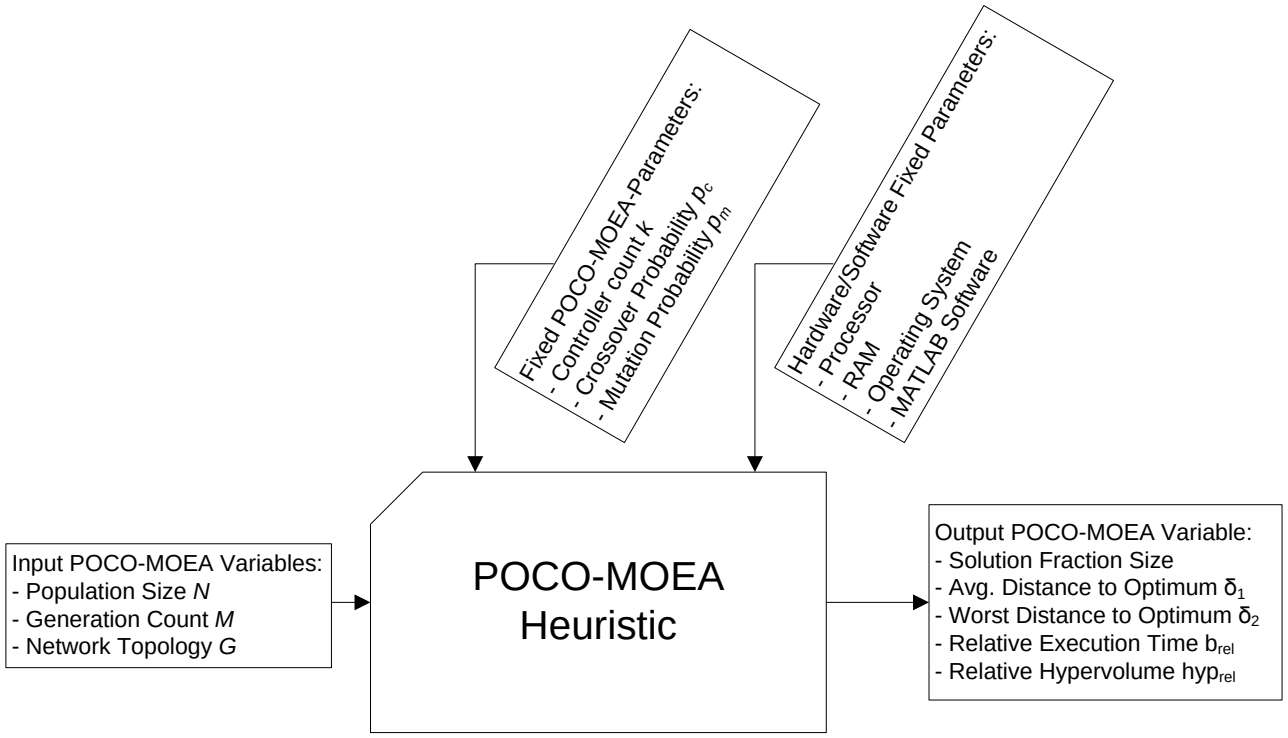

Figure 9. POCO-MOEA System Under Test (SUT) Representation

4.2.1 Assumptions.

The differing values for the population size are to measure how well the different population counts provide solutions which are closer to the pareto front. The generation count values determine how many new populations are developed. In general, it is assumed that more generations should result in a population closer to the optimal pareto front built by the enumerative solution. This should result in smaller d_1, d_2, and smaller differences between the HVs of the exhaustive and heuristics searches. Likewise, a larger population N should result in a solution set with more opportunities to come closer to the optimum pareto front. The different crossover probability determines how often solutions are swapped between parents. In general, a higher crossover probability should result in a more diverse final population that covers more of the Pareto front.

4.2.2 Tested Networks.

The static network topologies for performance evaluations are pulled from a database of open source network topology files made available at the Internet Topology Zoo [10]. Multiple topologies of different sizes are used in this scenario; they are further explained later in this section. Multiple network sizes and layouts are used to provide a comparative guideline of performance between the algorithms with respect to the various metrics (explained in Section 4.2.4).

The Test network in Figure 10 is the sample network created for the original POCO and demonstrated in the associated heuristics papers [8, 39, 40]. It is a smaller network diagram that provides good connectivity between most nodes, allowing for a wide range of potential placements. Test network uses the .topo.mat file format.

The Roedunet network in Figure 11 provides a slightly larger network size with a topology layout leaning towards a 'hub-and-spoke' network design. It is believed this

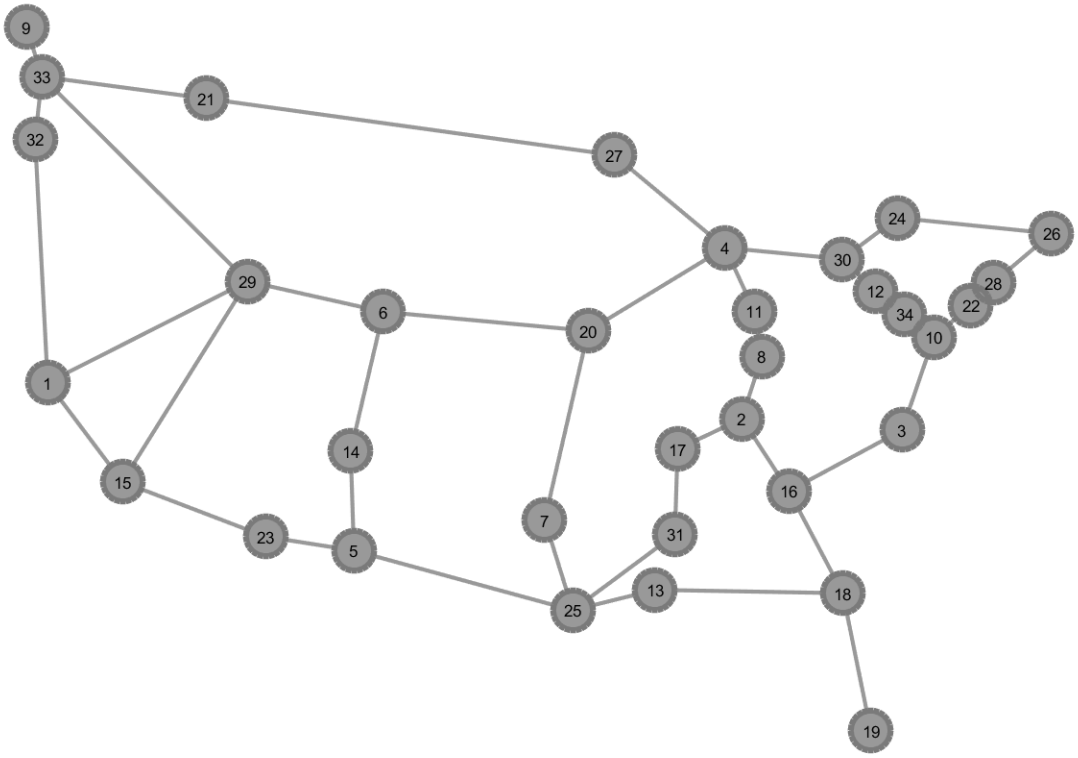

Figure 10. Test Network Diagram

network layout, with it's more centralized nodes, may have an impact on some of the metrics (explored on in Section 4.2.4). Roedunet uses the GraphML file format.

The Missouri network in Figure 12 is an even larger network with a very asymmetrical design. This network was chosen to be an overall larger network with some similarities to the Test Network. Missouri Network uses the GraphML file format.

Lastly, the Latnet network illustrated in Figure 13 is a still larger network with more centralized topology, similar to that of the Roedunet network. Latnet Network uses the GraphML file format.

4.2.3 System Hardware Specifications.

The experiments are run on a dedicated server with the following specifications:

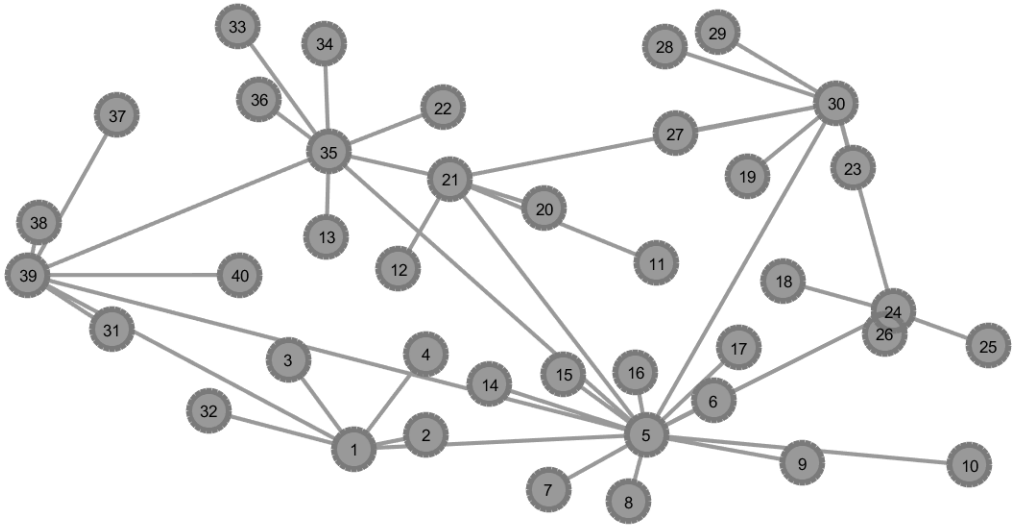

Figure 11. Roedunet Network Diagram

- Processor: AMD Opteron 4180 (2.6 GHz, 6 cores)
- Memory: 64 GB RAM
- OS Version: Windows 10
- Software Version: MATLAB 2015b

4.2.4 Metrics.

MOEAs have a variety of metrics available for comparing performance between the Pareto fronts generated by heuristic solutions, known as P_{known}. Some require having the knowledge of the original exhaustive search (P_{true}). Several of these are used for measuring POCO-MOEA performance.

The first metric is fraction of the solution space. This is simply a fractional

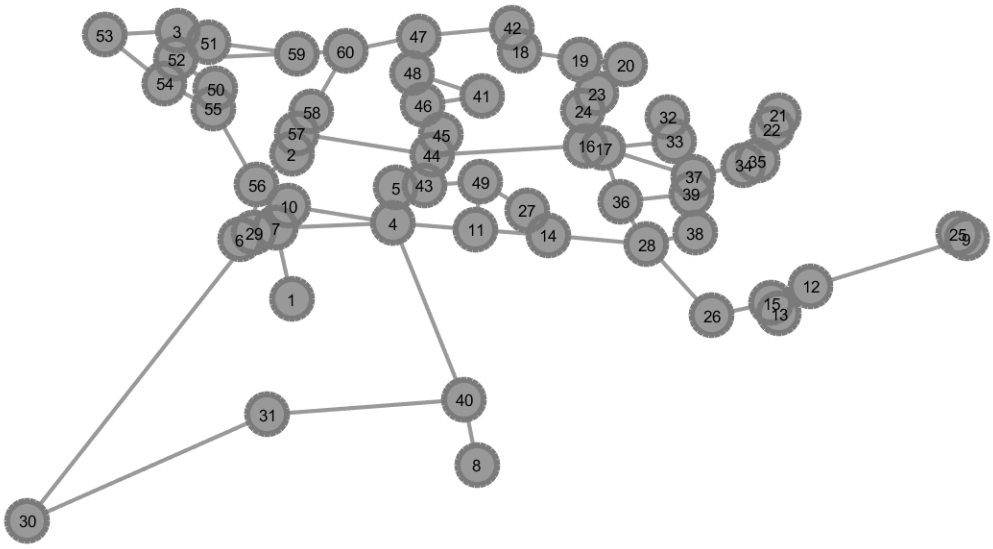

Figure 12. Missouri Network Diagram

value of the size of the solution set presented by POCO-MOEA compared to that of exhaustive search. The primary benefit of this metric is the amount of data saved by using a smaller set of data points for a heuristic search.

Several other metrics which apply to MOEAs are summarized in Dr. Coello Coello's book on EAs [6]. Two of the more popular metrics are:

1. **Hypervolume (HV):** This metric measures the volume of space generated by the points of a given Pareto front based on a selected reference point. The reference point is chosen by the user. Most commonly, this metric is used by comparing the respective HV values between P_{known} and P_{true} when using the same reference point for each.
2. **Generational Distance (GD):** This metric measures the distance between

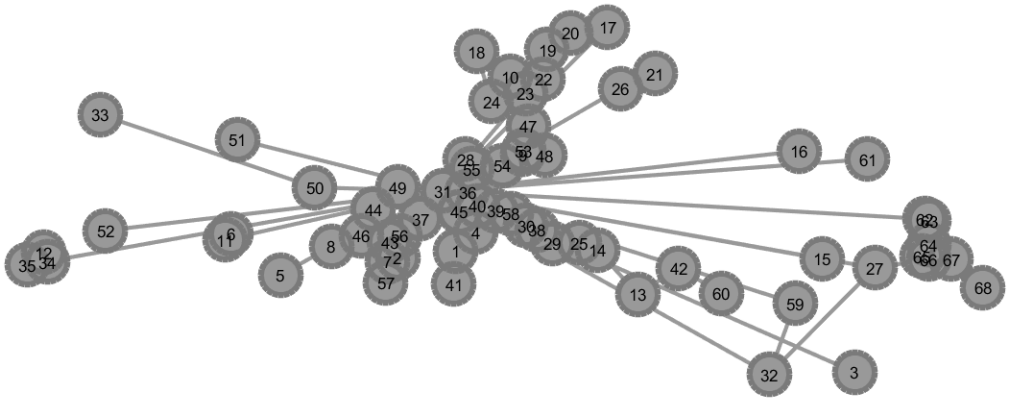

Figure 13. Latnet Network Diagram

the points generated P_{known} and the closest point from the optimal Pareto front, P_{true}. As the default POCO Framework performs an exhaustive search for controller placement, it is possible to generate an optimal Pareto front based on the scenarios given in Section 2.5.8.1.1.

Retrieving values for each of these metrics provides their own challenges. In the case of HV, the primary difficulty comes from the wide range between values in the objective functions. This makes it more difficult to come up with a reliable value for HV that can be compared between individual networks. This is addressed in POCO-MOEA by normalizing the values for all objective functions between 0 and 1. Because of this, it is easier to calculate the Hypervolume value, which is based on the objective function values, by setting the reference points to all 1s for each objective function.

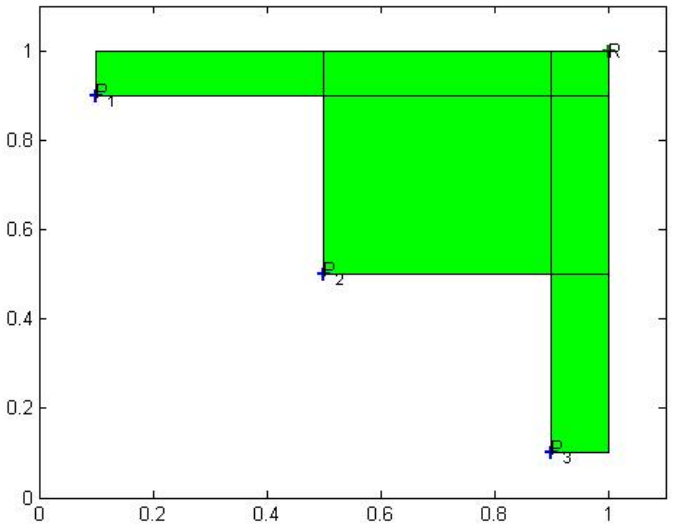

Figure 14. 2-Dimensional Hypervolume Example [44]

A 2-Dimensional example of this is shown in Figure 14. The point labeled 'R' on the top right of the graph is a $(1, 1)$ reference point, while P_1, P_2 and P_3 are members of a pareto front. The further the pareto front is away from the reference point, the larger the HV. In this case, a larger HV is better since each objective function is a minimization function. Having a pareto front value larger than the reference point would be an invalid solution in this case.

To determine how close the HV of each heuristic experiment comes to the optimum HV, a metric called hyp_{rel} measures the fraction of the heuristic HV hyp_{heu}, over the exhaustive HV, hyp_{exh}. This gives the final formula $hyp_{rel} = \frac{hyp_{heu}}{hyp_{exh}}$. A value of 1 would mean the heuristic HV matches the exhaustive HV exactly.

There is one concern to address with the way the HVs are calculated for these experiments. Because computing the HV exactly can be a time consuming process, the function used to compute the HV for each solution set is an estimate. A number of points are generated at random between the origin and the n-dimensional reference point. The HV is calculated as the percentage of points that are dominated by the

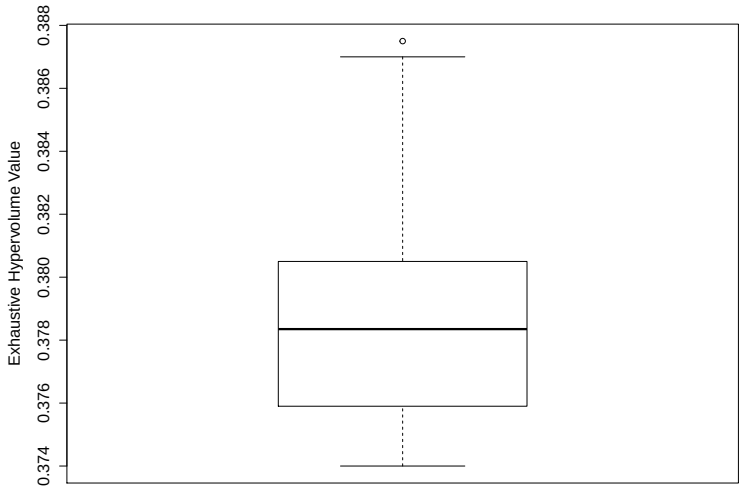

Figure 15. Exhaustive Hypervolume Value Boxplot for Test Network

solutions pareto front. Because every measurement of HV generates a new set of randomly distributed points, every measurement of HV is slightly different. This is reflected in the HV boxplot in Figure 15, which represents the hypervolume value based on 30 evaluations. The range of these HV values is 1.35%. The mean of these HVs is used as the hyp_{exh} value. A t-test shows this value is within a 95% confidence interval.

While GD specifically is not being used as a metric for this experiment, a very similar series of metrics is used by the authors of the PSA heuristic [39] which was borrowed from the original paper for PSA [45]. These metrics are borrowed for the purposes of evaluating POCO-MOEA and are the following:

- δ_1 - Measures the average distance from a pareto front solution generated by a reference set, R, to a pareto front solution by an approximation set, M, and is defined as

$$\delta_1 = \frac{1}{|R|} \sum \left\{ \min_{x \in M} \{c(\mathbf{x}, \mathbf{y})\} \right\} \tag{4.2.1}$$

- δ_2 - Measures the maximum distance from a pareto front solution generated by a reference set, R, to a pareto front solution by an approximation set, M

$$\delta_2 = \max_{y \in R} \left\{ \min_{x \in M} \{c(\mathbf{x}, \mathbf{y})\} \right\} \tag{4.2.2}$$

These values are based on the following equations, and are computed by the file paretoDist.m, developed by the team of the original POCO program. In the δ equations, the term $c(\mathbf{x}, \mathbf{y})$ refers to the following objective:

$$c(\mathbf{x}, \mathbf{y}) = \max_{j=1,\ldots,J} \{0, w_j(f_j(\mathbf{y}) - f_j(\mathbf{x})\} \tag{4.2.3}$$

where $j \in J$ is the number of objectives, and seeks to find the maximum value between 0 or the distance between objective vectors $\mathbf{x}$ and $\mathbf{y}$. A resultant value of 0 would mean that the two objective vectors are equivalent.

The last metric used is relative time elapsed between exhaustive and POCO-MOEA experimental runs. When measuring time elapsed as an absolute statistic, results can vary widely depending on the hardware the modeling software is run on, leading to unreliable metric data. To produce a metric that is not hardware dependent, the time elapsed is given as a fraction between the execution time of the POCO-MOEA heuristic, b_{heu}, over that of the exhaustive run, b_{exh}. This resulting value is given as $b_{rel} = \frac{b_{heu}}{b_{exh}}$. A b_{rel} value of 1 means that the heuristic takes the same amount of time to complete a task as the exhaustive solution. Being able to produce a small b_{rel} value is important in the performance evaluation of any heuristic, and POCO-MOEA is no exception.

All that said, the final list of metrics is shown in Table 9.

To investigate the reliability of these values, the mean and standard deviations are provided for each metric, at each parameter setting of every tested network.

Table 9. Metrics Measured for the Performance of POCO-MOEA

Symbol	Metric	Purpose
N/A	Fractional Solution Size	Measures fractional size of POCO-MOEA solution space compared to solution space of exhaustive search
δ_1	Delta 1	Measures average distance from solution space point on heuristic pareto front to the closest solution space point on exhaustive pareto front
δ_2	Delta 2	Measures worst case distance from solution space point on heuristic pareto front to the closest solution space point on exhaustive pareto front
b_{rel}	Relative Execution Time	Measures fractional time difference between solution generation by the heuristic and exhaustive search algorithms
hyp_{rel}	Relative Hypervolume	Measures fractional difference in the HVs generated between the heuristic and exhaustive search pareto fronts

4.3 Summary

This chapter reviewed the goals of the thesis, then covered the design of the experiment and variable parameters being tested. The metrics used for measuring POCO-MOEA performance are also explained

V. Results and Analysis

In this chapter the performance metrics detailed in Chapter IV are analyzed and evaluated between the different algorithms. The results for each network is presented as its own section, with each metric for each network given in subsections. The nature of network topologies having an impact on metric values means that metric values are only indicative of an individual network.

5.1 Metric Analysis

Coding efficiency played a significant role in the speed of certain algorithms. In particular, the speed of the POCO-MOEA algorithm was heavily impacted based on the code used for genetic operators. Certain components, such as searching for pareto solutions, could be done through the use of either the 'repmat' or the 'ismember' function. Code testing revealed a nearly 10-fold increase in computation speed when repmat is used in place of ismember. Even when solutions are calculated in speeds on the order of seconds, such speed improvements are significant. Thus, the efficiency of the code can have a significant impact on the b_{rel} values provided in the results. While significant time was spent optimizing the code for POCO-MOEA, there are definitely further modifications that could be made to the algorithm to further improve performance.

In order to pull the necessary information from the results generated by the experiments, the data is saved in a data structure to be read and interpreted by the R programming language. To accomplish this, the data is saved in version '-7.3' of the '.mat' file, or the Hierarchical Data Format, Version 5 (HDF5). The data is then read into R using the CRAN 'h5' package [46]. This brought the results data into a suitable format for the purposes of performing means and SD testing, as well as

generating boxplots.

All of the boxplots are in linear scale with the exception of the b_{rel} metric, which is plotted in logarithmic values.

5.1.1 All Networks.

5.1.1.1 Fraction of Search Space.

Because the purpose of an MOEA is to save time by limiting the number of solutions explored compared to an enumerative search, an important metric is the size of the heuristic solution set compared to that of the solution space. If the value of N is unchanged, the fraction of the solution space searched decreases as the size of the network increases. This has the potential for both positive and negative ramifications. Searching a smaller fraction of the solution space can provide a significant time saving (improving b_{rel}), but those savings can be lost if the smaller set of solutions produces poor results (given in larger values for δ_1, δ_2, and hyp_{rel}). These fractions are presented in Table 10. Note that this metric only represents how much of the available solution space is presented by POCO-MOEA with certain parameter settings for N, not how much of the solution space is actually searched over the course of POCO-MOEA.

Table 10. Fraction of Search Space Presented for Each POCO-MOEA Population Size

Network	# of Nodes	Size of Exhaustive Search Space (k=5)	Fraction of Search Space when N =		
			200	800	8000
Test	34	278,256	7.1876e-4	2.8751e-3	2.8751e-2
Roedunet	39	575,757	3.4737e-4	1.3895e-3	1.3895e-2
Missouri	60	5,461,512	3.6620e-5	1.4648e-4	1.4648e-3
Latnet	68	10,424,128	1.9186e-5	7.6745e-5	7.6745e-4

5.1.2 Test Network.

5.1.2.1 δ_1 Metric.

The δ_1 plot for the test network is given in Figure 16. A clear progression can be seen as the number of generations increases, the value decreases. This indicates points on the heuristic pareto front gradually come closer to those of the exhaustive pareto front. In all cases however, progressing beyond 50 generations seemed to produce very little benefit. In fact, in the case of the small population size (N=200) the mean of the δ_1 value improved by 4 ten-thousandths of a point between $M = 50$ and $M = 100$.

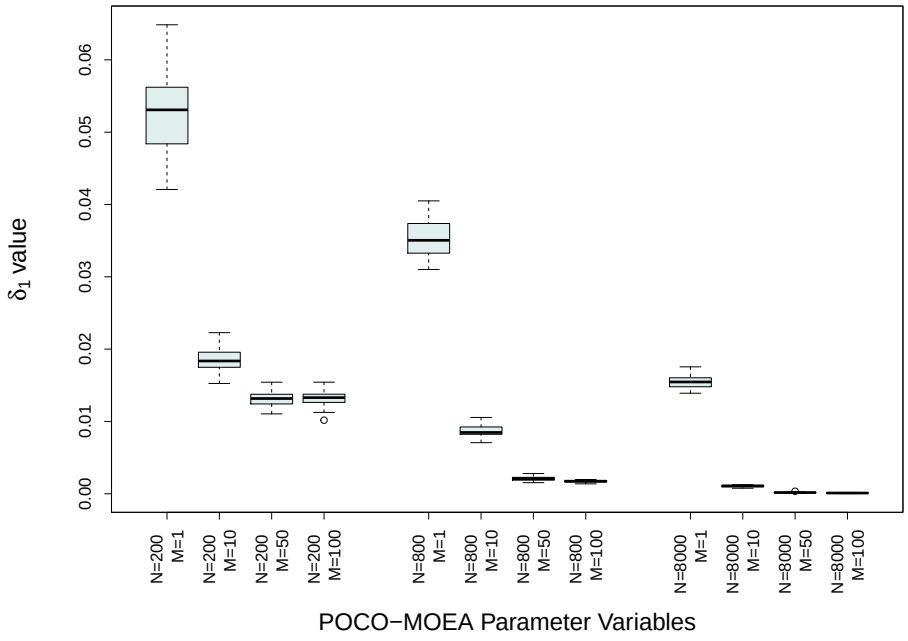

Figure 16. Test Network δ_1 Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

The data shows that the extra computational effort and time expended by the additional MOEA operators is more of a hindrance than a help. Additionally, for the same number of generations, increasing the population size also improves the δ_1 value. The representative means and standard deviations are displayed in Table 11. The standard deviation decreases rapidly once M increases beyond one.

Table 11. Statistics of δ_1 Values for Test Network

N	M	Mean	Std. Dev.
200	1	5.2582e-2	5.6390e-3
200	10	1.8514e-2	1.8476e-3
200	50	1.3221e-2	9.7421e-4
200	100	1.3180e-2	1.1118e-3
800	1	3.5323e-2	2.5169e-3
800	10	8.6594e-3	7.4145e-4
800	50	2.1012e-3	3.2697e-4
800	100	1.7336e-3	1.5236e-4
8000	1	1.5522e-2	9.4596e-4
8000	10	1.0569e-3	1.2847e-4
8000	50	1.6980e-4	5.9734e-5
8000	100	1.0464e-4	3.9553e-4

5.1.2.2 δ_2 Metric.

Plots from the Test Networks' δ_2 metric behave in much the same manner as those of δ_1, as demonstrated in Figure 17. Namely, increasing the number of generations improves the δ_2 value, as does increasing the population size. One difference between this figure and Figure 16, however, is that increasing the number of generations beyond 50 produces further improvements for the population sizes 200 and 800, even though 8000 remained relatively unchanged. The table of Means and SDs are in Table 12. Values behave similar to that of the δ_1 table.

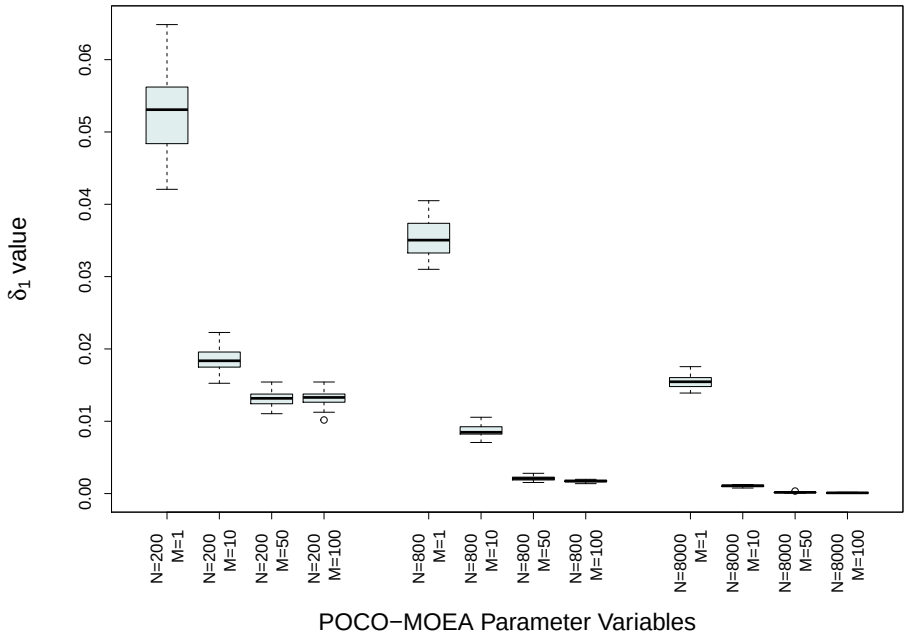

Figure 17. Test Network δ_2 Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 12. Statistics of δ_2 Values for Test Network

N	M	Mean	Std. Dev.
200	1	1.6065e-1	2.6500e-2
200	10	9.3243e-2	1.7094e-2
200	50	6.5544e-2	7.2883e-3
200	100	5.8199e-2	6.6722e-3
800	1	1.2186e-1	1.8492e-2
800	10	6.5595e-2	1.2334e-2
800	50	5.0366e-2	1.5319e-2
800	100	3.4369e-2	9.7644e-3
8000	1	8.1977e-2	1.0923e-2
8000	10	3.2600e-2	6.8510e-2
8000	50	2.3861e-2	7.1513e-3
8000	100	2.2939e-2	7.4439e-3

5.1.2.3 b_{rel} Metric.

The b_{rel} boxplots in Figure 18 demonstrate an interesting trend in amount of time taken to produce the heuristic solution when compared to the exhaustive solution. The dashed line across the x-axis represents the point in which it takes the same amount of time to perform POCO-MOEA as the exhaustive search. In this case, since we are dealing with a smaller network, it does not take much effort in order to surpass this threshold, as we are dealing with a limited number of potential solutions for exhaustive search. In such a case, the extra computation cost invoked by performing the MOEA operators exceeds that of simply performing the exhaustive search. The means and SDs are in Table 13. Because the logarithmic plot of b_{rel} can be misleading, the table provides more precise values for execution time.

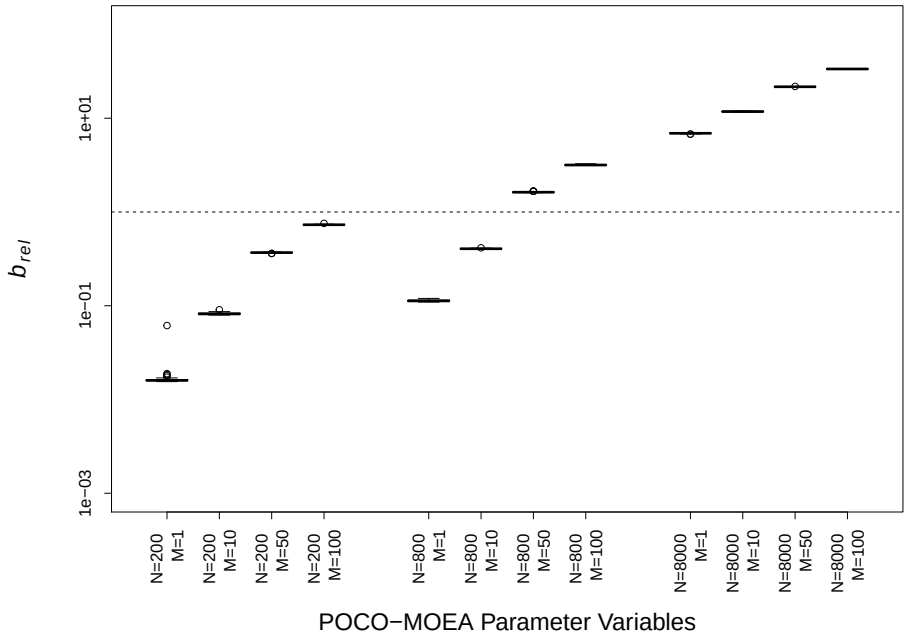

Figure 18. Test Network b_{rel} Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 13. Statistics of b_{rel} Values for Test Network

N	M	Mean	Std. Dev.
200	1	1.7803e-2	8.2884e-3
200	10	8.2454e-2	2.2628e-3
200	50	3.6834e-1	3.0536e-3
200	100	7.3145e-1	6.4795e-3
800	1	1.1345e-1	2.9147e-3
800	10	4.0546e-1	3.5228e-3
800	50	1.6289e+0	1.7250e-2
800	100	3.1848e+0	3.6613e-2
8000	1	6.9045e+0	6.1993e-2
8000	10	1.1780e+1	1.040e-1
8000	50	2.1649e+1	6.4938e-2
8000	100	3.3508e+1	8.0812e-2

5.1.2.4 hyp_{rel} Metric.

The HV metric boxplots given in Figure 19 provide probably the most telling information of all in regards to improvement as N and M increase. In comparison to the δ_1 and δ_2 values, the HV values continue to improve between the M values of 50 and 100. The exception to this is the case when N=8000, where the HV fraction is already equal to 1 for M=10 or higher. The table of statistics provided in Table 14 shows very minor improvements in the hyp_{rel} value as generations increase, with the exception of $N = 8000$.

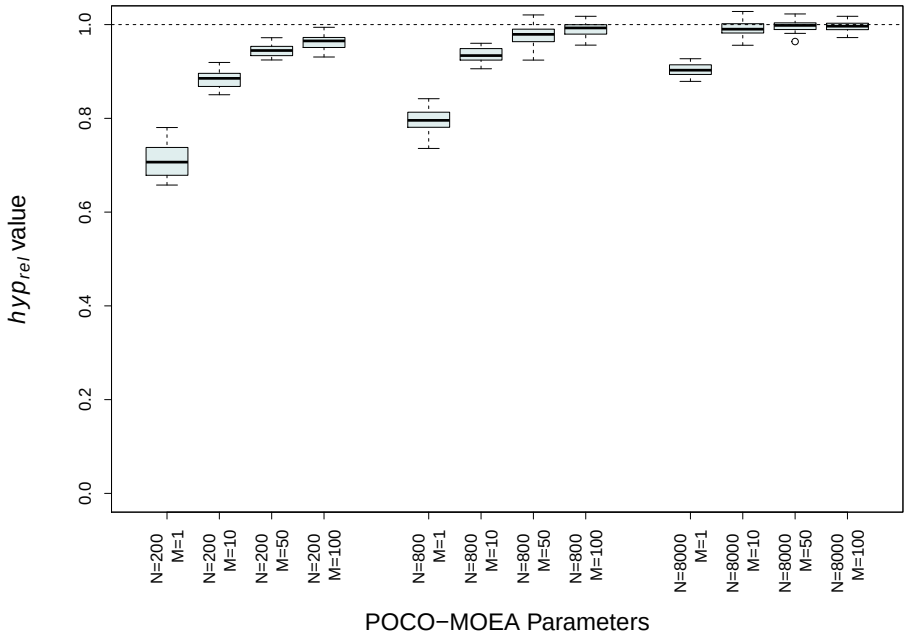

Figure 19. Test Network hyp_{rel} Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 14. Statistics of hyp_{rel} Values for Test Network

N	M	Mean	Std. Dev.
200	1	0.7093	3.3101e-2
200	10	0.8848	1.7958e-2
200	50	0.9449	1.3555e-2
200	100	0.9634	1.5942e-2
800	1	0.7972	2.1986e-2
800	10	0.9356	1.4815e-2
800	50	0.9773	2.0806e-2
800	100	0.9908	1.4136e-2
8000	1	0.9028	1.3654e-2
8000	10	0.9909	1.6767e-2
8000	50	0.9966	1.1795e-2
8000	100	0.9963	1.1470e-2

5.1.3 Roedunet Network.

5.1.3.1 δ_1 Metric.

In the case of the Roedunet network, the δ_1 boxplots presented in Figure 20 provide interesting information. The δ_1 values for this network are almost universally higher than those of other networks. An untested theory as to the cause behind this is the topology of the network itself, with its very strong 'hub-and-spoke' layout. Because there are clearly defined centralized 'hubs' existing in the network, not choosing these hubs as part of a solution would result in significantly worse values for some of the solution metrics, particularly average node-to-controller latency and worst case node-to-controller latency. These in turn would raise the δ_1 and δ_2 values. Also noted is how at higher M values, even solutions with N=200 quickly converge to nearly optimal values. The means and SDs are available in Table 15, where the SD values demonstrate how much tighter the δ_1 value becomes as M increases.

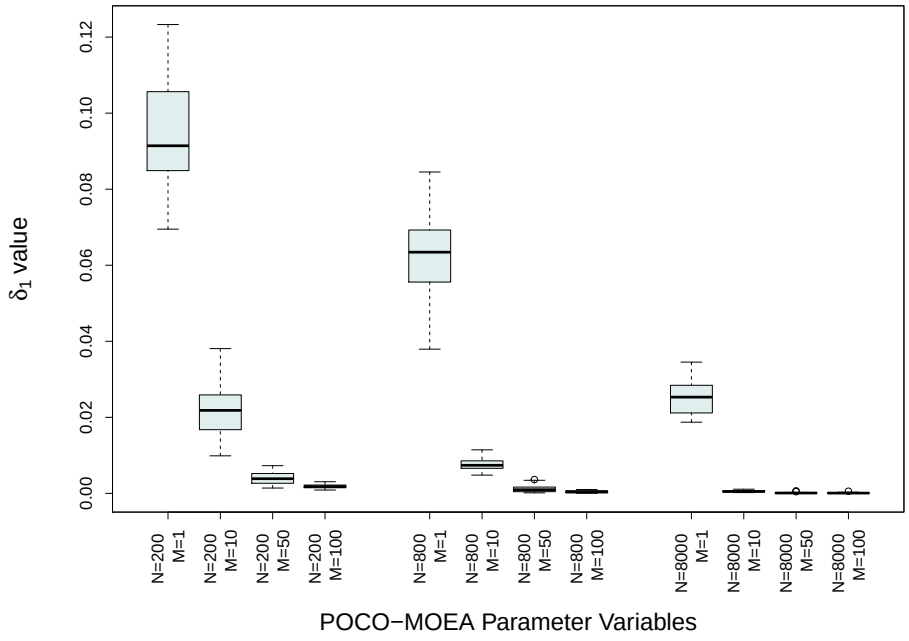

Figure 20. Roedunet Network δ_1 Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 15. Statistics of δ_1 Values for Roedunet Network

N	M	Mean	Std. Dev.
200	1	9.4671e-2	1.5457e-2
200	10	2.2172e-2	6.7691e-3
200	50	4.0626e-3	1.5848e-3
200	100	1.8810e-3	4.7588e-4
800	1	6.2164e-2	1.2642e-2
800	10	7.7238e-3	1.6397e-3
800	50	1.2009e-3	9.5111e-4
800	100	4.4449e-4	2.7879e-4
8000	1	2.5502e-2	4.4620e-3
8000	10	5.6352e-4	2.5511e-4
8000	50	1.2824e-4	1.3948e-4
8000	100	9.9888e-5	1.1322e-4

5.1.3.2 δ_2 Metric.

Similar to Figure 20, the δ_2 boxplot in Figure 21 has demonstrably higher values compared to the other networks in this experiment at low M. As M increases however, these values begin to converge to be more in line with those of the other networks. Further testing is required to determine the cause of these discrepancies, although it is believed that the network topology plays a significant role. The means and standard deviations in Table 16 display similar behavior to those in Table 15.

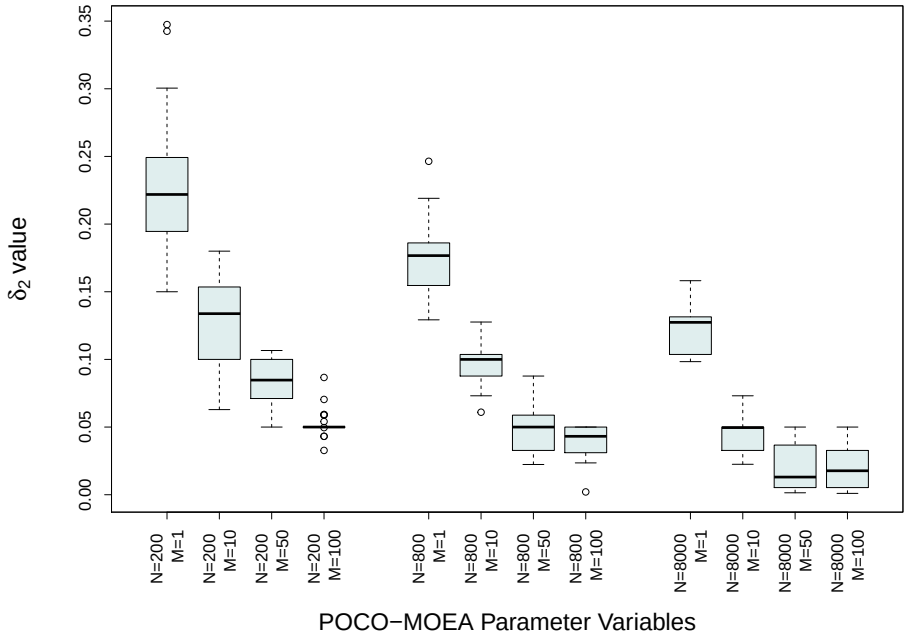

Figure 21. Roedunet Network δ_2 Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 16. Statistics of δ_2 Values for Roedunet Network

N	M	Mean	Std. Dev.
200	1	2.2854e-1	5.0745e-2
200	10	1.3087e-1	2.6958e-2
200	50	8.3064e-2	1.7731e-2
200	100	5.1903e-2	8.9210e-3
800	1	1.7639e-1	2.7307e-2
800	10	9.7200e-2	1.5905e-2
800	50	5.1143e-2	1.7392e-2
800	100	3.9051e-2	1.1996e-2
8000	1	1.2386e-1	2.0449e-2
8000	10	4.3530e-2	1.3351e-2
8000	50	2.0044e-2	1.7041e-2
8000	100	1.9240e-2	1.6198e-2

5.1.3.3 b_{rel} Metric.

The b_{rel} values for Roedunet appear in Figure 22. Similar to the Test Network, we see that after a certain amount of computational effort it actually takes more time to perform the heuristic than it does the exhaustive search. This lends more credence to the previous evidence that POCO-MOEA is not well suited for smaller networks. Table 17 lends further evidence to this fact; in the higher values of N and M, the b_{rel} approached and even exceeds 1. This is not a desirable trait for any heuristic. This provides further evidence that the use of an MOEA provides the largest benefits when the effects of an NP-hard problem become more obvious (when the solution space is large).

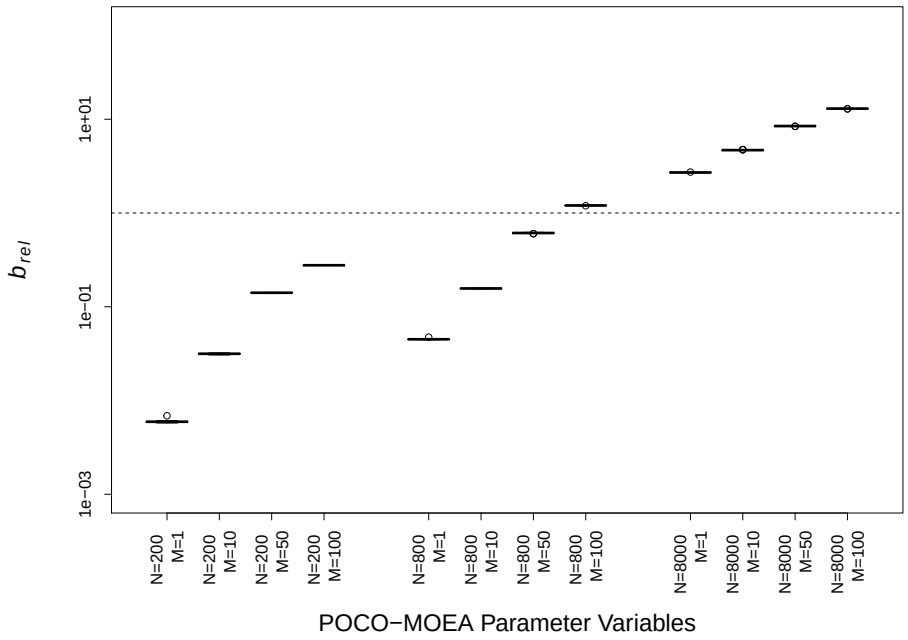

Figure 22. Roedunet Network b_{rel} Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

5.1.3.4 hyp_{rel} Metric.

The relative HV boxplots for Roedunet are shown in Figure 23. Even at low N, values are nearly in line with that of exhaustive search are achieved. Again, a theory

Table 17. Statistics of b_{rel} Values for Roedunet Network

N	M	Mean	Std. Dev.
200	1	5.9610e-3	1.9202e-4
200	10	3.1514e-2	4.0658e-4
200	50	1.4105e-1	6.6161e-4
200	100	2.7714e-1	9.5683e-4
800	1	4.5095e-2	5.3477e-4
800	10	1.5670e-1	7.6091e-4
800	50	6.1212e-1	4.3371e-3
800	100	1.2030e+0	3.6611e-3
8000	1	2.7046e+0	9.4222e-3
8000	10	4.6838e+0	2.2388e-2
8000	50	8.4474e+0	2.5114e-2
8000	100	1.2980e+1	3.3694e-2

behind this is the strong hub-and-spoke topology of the network, which would lend itself strongly to a subset of solutions (those which choose hub nodes as controllers) to be significantly better than others. Table 18 shows how even the $N = 200$ population can get within 1% of the optimum HV when the generation count is high. Increasing the values for N allows hyp_{rel} to come within 1% with even lower M values, although this also introduces a balancing act with the trade-offs made for b_{rel}.

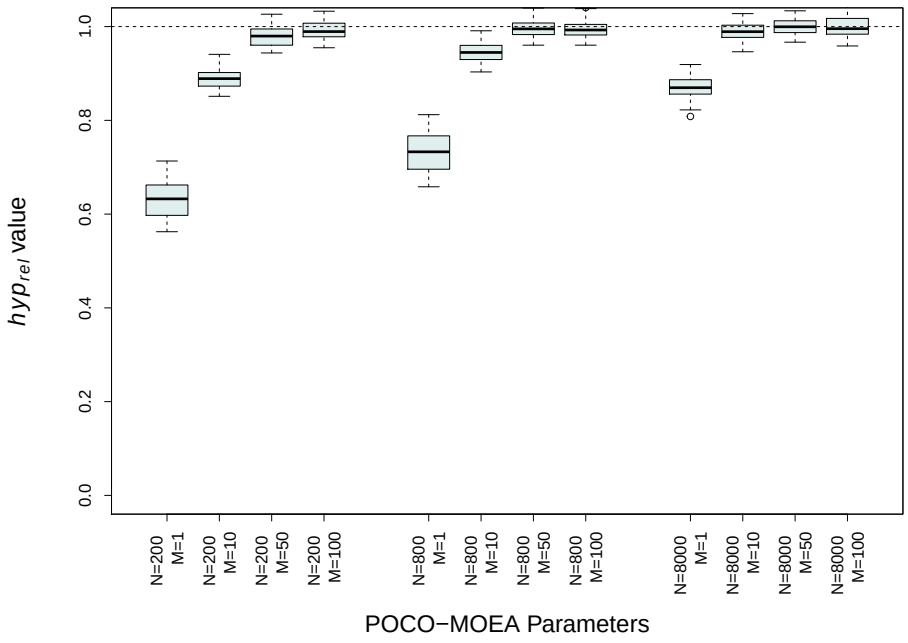

Figure 23. Roedunet Network hyp_{rel} Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 18. Statistics of hyp_{rel} Values for Roedunet Network

N	M	Mean	Std. Dev.
200	1	0.6315	3.9840e-2
200	10	0.8899	2.3653e-2
200	50	0.9796	2.1940e-2
200	100	0.9912	2.0329e-2
800	1	0.7336	4.2813e-2
800	10	0.9459	2.2792e-2
800	50	0.9964	1.8847e-2
800	100	0.9944	1.9488e-2
8000	1	0.8704	2.5502e-2
8000	10	0.9899	1.9521e-2
8000	50	0.9992	1.6505e-2
8000	100	0.9992	2.1710e-2

5.1.4 Missouri Network.

5.1.4.1 δ_1 Metric.

The δ_1 boxplots for the Missouri network are given in Figure 24. These plots provide information very similar to that of the Test network; namely, that increased M improves the value up to a certain threshold, depending on N. That threshold then further improves with increased N values. The population size of N=8000 appears to produce near optimum values. The means for $N = 200$ and 800 as given in Table 19 show that increasing M in certain instance can result in a worse result. Fine-tuning the M value can become necessary and may be dependent on the network being surveyed.

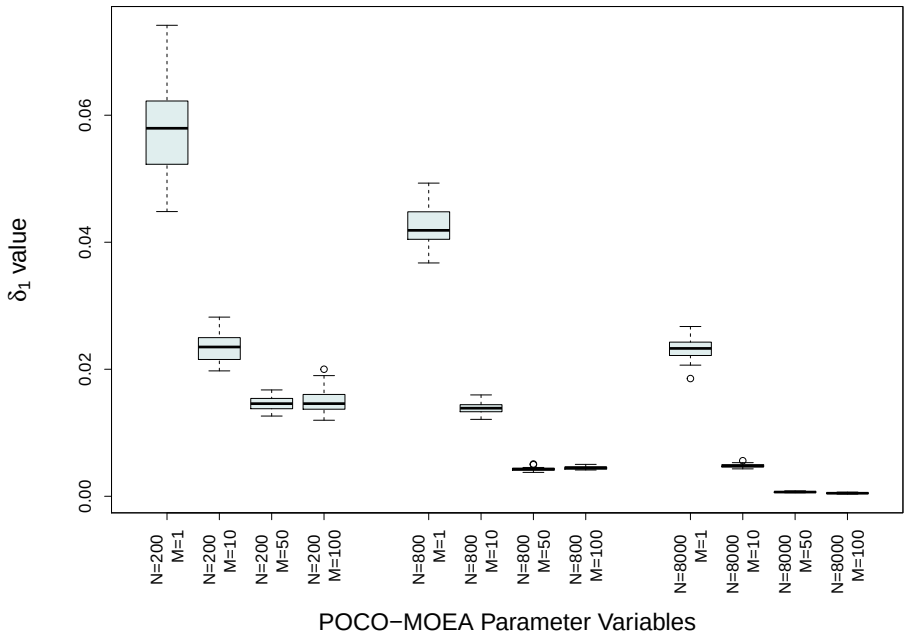

Figure 24. Missouri Network δ_1 Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 19. Statistics of δ_1 Values for Missouri Network

N	M	Mean	Std. Dev.
200	1	5.7660e-2	6.9391e-3
200	10	2.3596e-2	2.4164e-3
200	50	1.4590e-2	1.0627e-3
200	100	1.5031e-2	1.9347e-3
800	1	4.2591e-2	3.3962e-3
800	10	1.3892e-2	8.8514e-4
800	50	4.2857e-3	2.7365e-4
800	100	4.4699e-3	2.1236e-4
8000	1	2.3178e-2	1.5398e-3
8000	10	4.8040e-3	2.8523e-4
8000	50	6.7271e-4	8.3272e-5
8000	100	5.0116e-4	7.9949e-5

5.1.4.2 δ_2 Metric.

Much like the δ_1 plot, the δ_2 boxplot in Figure 25 demonstrates very similar behavior. The means for $N = 200$ in Table 20 further demonstrate the dangers of using a higher M value than necessary. The fact that the value's improvement is very slight between $N = 800$ and $N = 8000$ is a curious case. It is believed that in most cases, increasing the population size of the heuristic should also increase the size of the heuristic pareto front. In theory, such an action should provide more data points that can approach closer to the pareto front of the exhaustive search. That this did not happen outside of a few outliers (seen at the $N = 8000/M = 50$ and $N = 8000/M = 100$ boxplots), warrants further investigation.

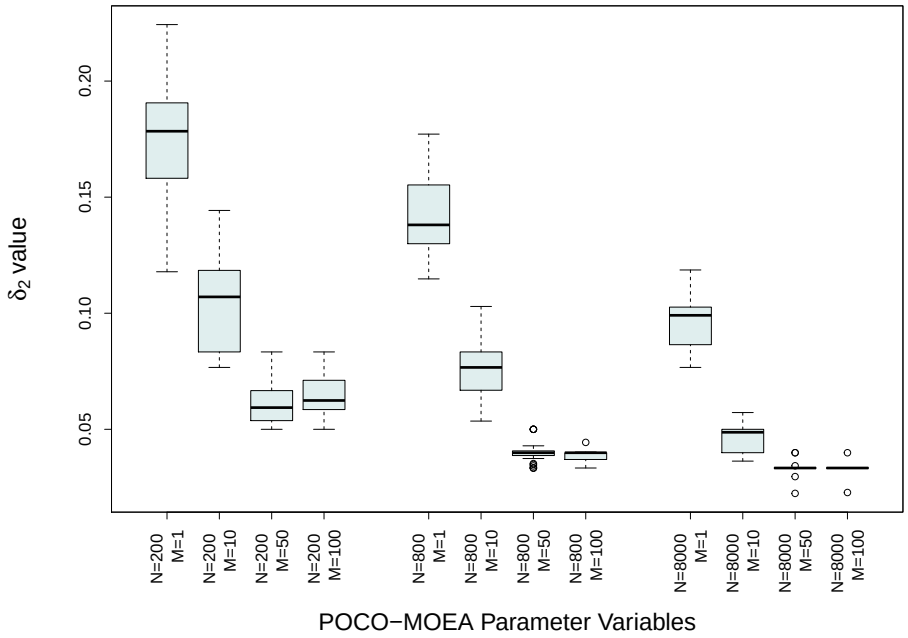

Figure 25. Missouri Network δ_2 Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 20. Statistics of δ_2 Values for Missouri Network

N	M	Mean	Std. Dev.
200	1	1.7373e-1	2.7739e-2
200	10	1.0639e-1	1.9609e-2
200	50	6.1390e-2	9.6290e-3
200	100	6.4478e-2	8.5678e-3
800	1	1.4176e-1	1.7943e-2
800	10	7.7197e-2	1.1432e-2
800	50	4.0139e-2	4.6737e-3
800	100	3.8548e-2	2.7191e-3
8000	1	9.6711e-2	1.1301e-2
8000	10	4.6203e-2	5.3713e-3
8000	50	3.3315e-2	2.7584e-3
8000	100	3.3201e-2	2.3058e-3

5.1.4.3 b_{rel} Metric.

The b_{rel} boxplots for Missouri in Figure 26 provides the first evidence that POCO-MOEA can save significant execution time over exhaustive search. With a solution space of over 5 million data points as given in Table 10, it becomes evident that this value continues to improve as network sizes increase. One thing to note in Table 21 is the extremely small standard deviation values, demonstrating the reliability of the heuristic runs' speed.

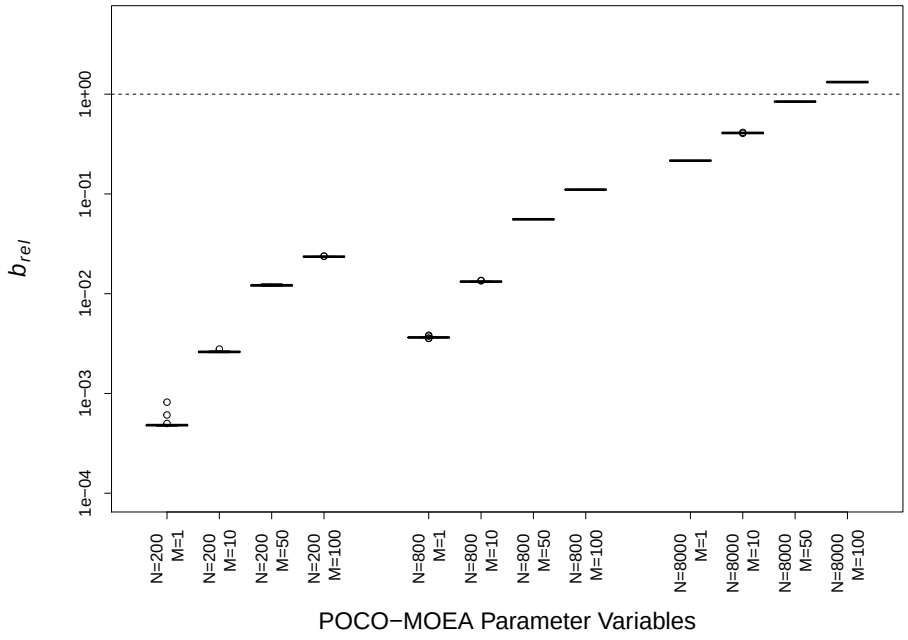

Figure 26. Missouri Network b_{rel} Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

5.1.4.4 hyp_{rel} Metric.

The boxplots for Missouri of hyp_{rel} in Figure 27 behave in a similar fashion to that of the Test network. The means and SD values provided in Table 22 show that even when the δ_1 and δ_2 values stagnate, the HV of the pareto front of solutions is still able to slightly improve as both N and M increase. Despite this however, the hyp_{rel} values are not able to approach as closely to 1 for this network as they do for

Table 21. Statistics of b_{rel} Values for Missouri Network

N	M	Mean	Std. Dev.
200	1	4.9619e-4	6.5412e-5
200	10	2.6189e-3	4.0976e-5
200	50	1.2137e-2	1.8039e-4
200	100	2.3515e-2	1.0829e-4
800	1	3.6489e-3	4.7000e-5
800	10	1.3246e-2	1.0761e-4
800	50	5.5693e-2	2.3099e-4
800	100	1.1049e-1	3.0115e-4
8000	1	2.1569e-1	5.5416e-4
8000	10	4.0890e-1	1.8277e-3
8000	50	8.4229e-1	2.7603e-3
8000	100	1.3218e+0	3.7585e-3

the Test and Roedunet Networks. A theory as to the cause is the size of the pareto front generated by exhaustive search. As the size of the solution space increases, it is believed the size of the pareto front of solutions also increases. As the fraction of the solution space given by POCO-MOEA decreases, it may become more difficult for the heuristic to accurately represent the full pareto front represented by exhaustive search. Another possible cause may be the lack of fidelity in hypervolume values due to the software used (explained in Section 4.2.4). Using a small number of data points to calculate hypervolume values may result in inaccuracy as the size of the solution space increases. This does not explain, however, why the δ_1 and δ_2 metrics are still able to show good results. Further testing is required to determine the actual cause(s).

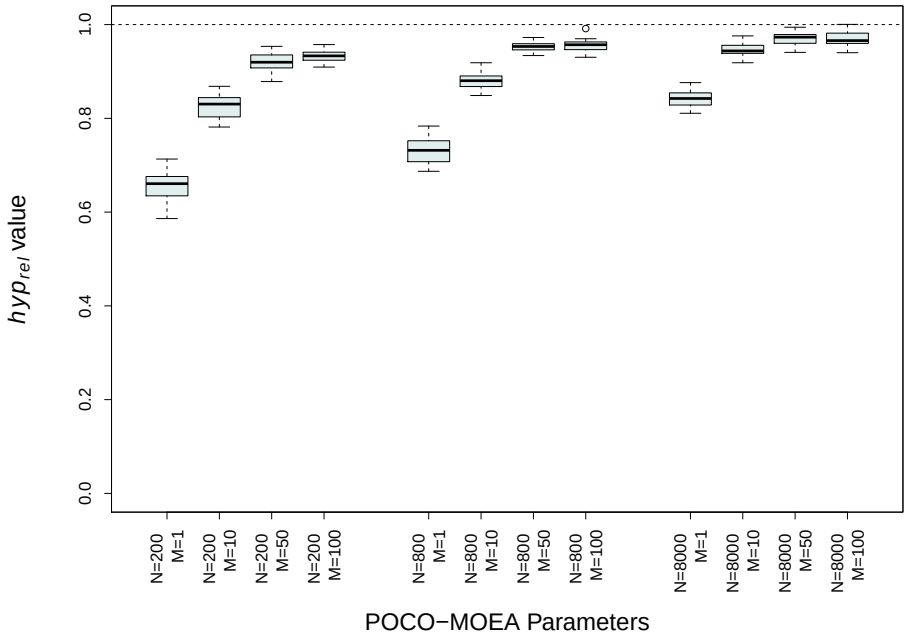

Figure 27. Missouri Network hyp_{rel} Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 22. Statistics of hyp_{rel} Values for Missouri Network

N	M	Mean	Std. Dev.
200	1	0.6567	3.0642e-2
200	10	0.8260	2.4457e-2
200	50	0.9202	1.8192e-2
200	100	0.9329	1.2515e-2
800	1	0.7305	2.6530e-2
800	10	0.8815	1.6863e-2
800	50	0.9529	9.5572e-3
800	100	0.9552	1.1963e-2
8000	1	0.8422	1.7322e-2
8000	10	0.9460	1.2798e-2
8000	50	0.9710	1.3291-2
8000	100	0.9698	1.5852e-2

5.1.5 Latnet Network.

5.1.5.1 δ_1 Metric.

As the largest network used for this set of experiments, Latnet provides the greatest insight as to the benefits of using POCO-MOEA over an exhaustive search. Starting with the δ_1 boxplots in Figure 28, it appears the size of N has less impact than the number of generations. As has been seen in the previous networks, an increase in M from 50 to 100 presents little to no improvement. A note for future research would be finding whether there is an M value lower than 50 which would still approach a threshold δ_1 value for a given N. Further experimentation is necessary to determine what that 'optimum' M value might be. Once again, this value is likely network dependent. The table of means and standard deviation in Table 23 also shows being able to achieve relatively small δ_1 values.

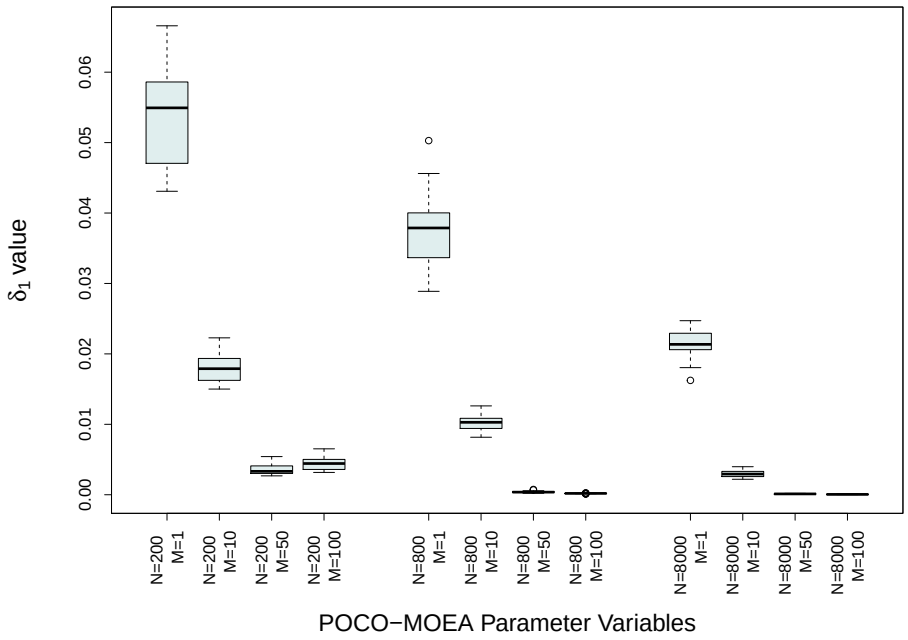

Figure 28. Latnet Network δ_1 Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 23. Statistics of δ_1 Values for Latnet Network

N	M	Mean	Std. Dev.
200	1	5.2560e-2	6.5027e-3
200	10	1.7939e-2	1.7928e-3
200	50	3.5930e-3	7.6077e-4
200	100	4.4740e-3	9.6635e-4
800	1	3.7271e-2	4.7385e-3
800	10	1.0311e-2	1.2252e-3
800	50	3.9729e-4	1.1652e-4
800	100	1.8696e-4	2.9187e-5
8000	1	2.1415e-2	1.9840e-3
8000	10	2.9789e-3	4.6936e-4
8000	50	1.1621e-4	4.2579e-5
8000	100	5.8925e-5	3.0683e-5

5.1.5.2 δ_2 Metric.

The δ_2 boxplots in Figure 29 produces few new surprises, only the high number of outliers for the scenario when N=200 while M=10. The boxplots for $N = 8000$, $M = 50$ and $N = 8000$, $M = 100$ seem to show that the δ_2 value does not really converge, but the standard deviation values given in Table 24 show that this solution spread does not increase much.

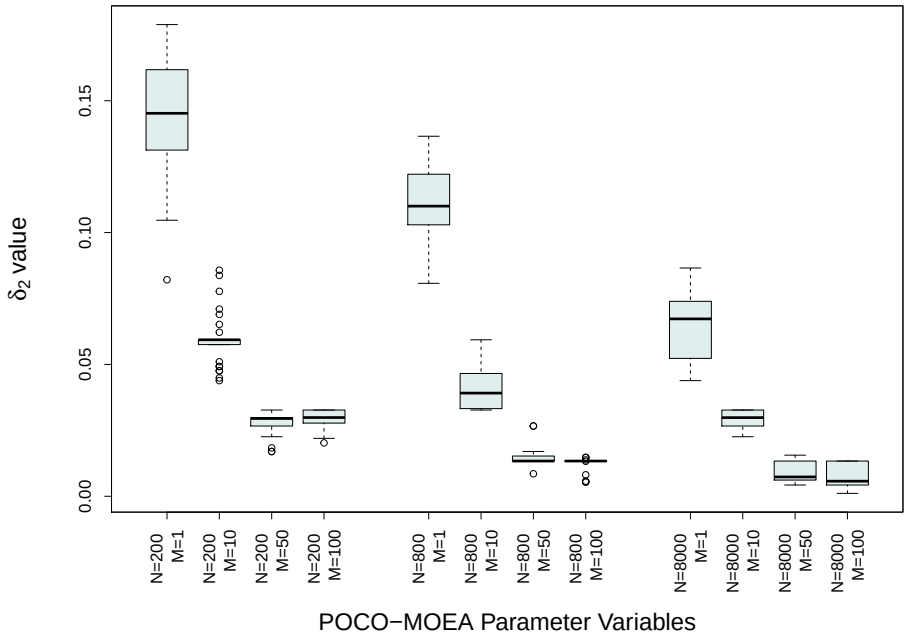

Figure 29. Latnet Network δ_2 Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Table 24. Statistics of δ_2 Values for Latnet Network

N	M	Mean	Std. Dev.
200	1	1.4176e-1	2.1669e-2
200	10	5.9757e-2	1.0004e-2
200	50	2.7682e-2	4.4064e-3
200	100	2.8971e-2	3.6289e-3
800	1	1.1153e-1	1.3931e-2
800	10	4.0383e-2	7.2072e-3
800	50	1.4940e-2	3.5773e-3
800	100	1.2545e-2	2.6135e-3
8000	1	6.4066e-2	1.2770e-2
8000	10	2.9391e-2	3.1611e-3
8000	50	8.9600e-3	3.7480e-3
8000	100	7.3782e-3	4.5811e-3

5.1.5.3 b_{rel} Metric.

For the b_{rel} boxplots for Latnet in Figure 30, it is shown that even for large values of N and M, an MOEA can still produce results in significantly less time than exhaustive search. The very low standard deviation values in Table 25 also shows how reliable the speed of these runs are even for larger networks. One curious exception, however, is the high frequency of slow outliers for $N = 200$ and $M = 1$. It is unknown as to the underlying cause for this wide variation. More experimentation is necessary.

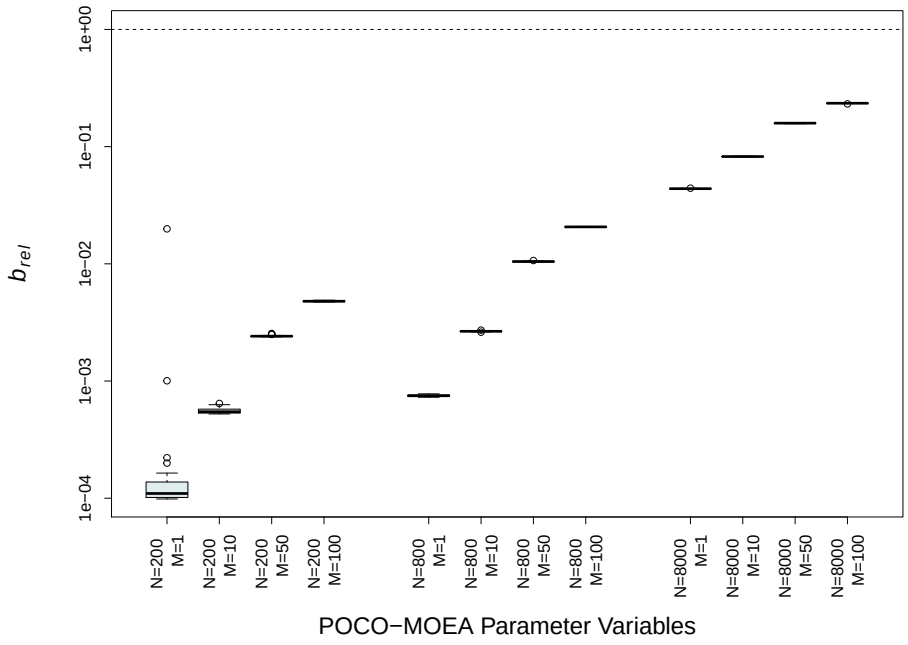

Figure 30. Latnet Network b_{rel} Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

5.1.5.4 hyp_{rel} Metric.

An interesting point of interest for the hyp_{rel} boxplots of the Latnet network in Figure 31 is that even with large values for N and M, POCO-MOEA is never quite able to approach a value of 1. This behavior is even more conspicuous than that shown by the hyp_{rel} heuristic for the Missouri network. Also similar to the Missouri

Table 25. Statistics of b_{rel} Values for Latnet Network

N	M	Mean	Std. Dev.
200	1	8.1082e-4	3.6061e-3
200	10	5.5851e-4	3.1862e-5
200	50	2.4263e-3	3.8535e-5
200	100	4.8050e-3	5.2744e-5
800	1	7.5069e-4	1.2721e-5
800	10	2.6591e-3	2.0879e-5
800	50	1.0477e-2	7.2044e-5
800	100	2.0671e-2	8.1562e-5
8000	1	4.3782e-2	1.7202e-4
8000	10	8.2349e-2	2.5318e-4
8000	50	1.5852e-1	5.9083e-4
8000	100	2.3446e-1	7.7714e-4

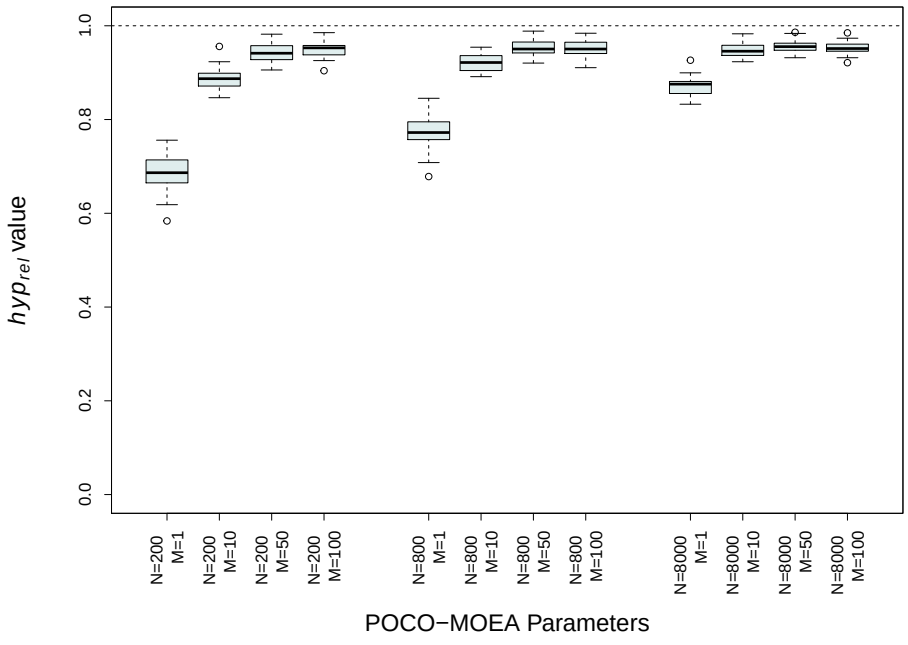

Figure 31. Latnet Network hyp_{rel} Heuristic Parameter Boxplot, $k = 5$, $p_c = 0.9$, $p_m = 0.2$

Network, the hyp_{rel} value seems to show little improvement as the M value increases

Table 26. Statistics of hyp_{rel} Values for Latnet Network

N	M	Mean	Std. Dev.
200	1	0.6864	4.0783e-2
200	10	0.8870	2.3584e-2
200	50	0.9420	1.9640e-2
200	100	0.9486	1.5668e-2
800	1	0.7737	3.5826e-2
800	10	0.9209	1.7995e-2
800	50	0.9522	1.5385e-2
800	100	0.9537	1.7500e-2
8000	1	0.8719	2.0541e-2
8000	10	0.9488	1.5079e-2
8000	50	0.9560	1.4269e-2
8000	100	0.9531	1.3861e-2

beyond 10, regardless of N value. Further testing is required to determine the underlying cause of the behavior for this network. The means values in Table 26 also show that increasing N produces little benefit.

5.2 Miscellaneous Data

Given in this section is additional data which is not part of the original hypothesis and goals, but is still deemed pertinent side information for the results. For the first metric, fraction of solution space used, the use of a heuristic drastically reduced the amount of solution space taken up by the final population at each experiment run. This creates the side benefit also greatly reducing the size of the data structures gathered by POCO-MOEA compared to that of exhaustive search; during testing, every experiment run using a heuristic produced a data structure taking up less than 500KB of disk space. In the case of smaller networks, this is not a huge difference. As network size increases however, the size of the solution set for exhaustive search also

increases rapidly (a full solution set for the Missouri network takes approximately 85MB of space; Latnet takes up 171MB). In early experimental tests with a 735 point network, a full solution set consumes over 8GB of space.

5.3 Summary

This chapter covered the results of the metrics for each network in turn, from the Test, Roedunet, Missouri and Latnet networks. The significance of the results of each metric for each network was also reviewed, and what it may mean for future research. Additional miscellaneous data not originally planned for review in Chapter IV were also covered.

VI. Conclusions and Future Work

In this thesis an enumerative program for determining the Pareto optimal set of solutions for the CPP is transformed into an MOEA program for solving the same problem, providing a vast reduction in solution computation time for larger networks in exchange for a slight reduction in the quality of the final solution. In this chapter the lessons learned about POCO-MOEA, and MOEAs in general, are discussed. Following that, future applications and potential projects of the use of MOEAs on the CPP are provided.

6.1 Review of Goals and Hypotheses

To restate, the hypothesis of this thesis is that the POCO-MOEA heuristic can produce a set of solutions which are nearly as optimal as those generated by exhaustive search when using a specified set of objective functions. In addition, it can do this with a much shorter execution time compared to exhaustive search; the original goal is at least 10 times faster. Lastly, it is theorized that the selected input parameters for POCO-MOEA have a significant impact on the performance of the heuristic. To accomplish this, the objectives of running data run tests on POCO-MOEA is to determine how altering two different input variables affects the execution time, as well as said execution time when compared to exhaustive search and closeness of the heuristic solution set to the optimum exhaustive solution set. To determine the validity of the goal and hypotheses, an experiment is built of running exhaustive search and a full-factorial set of heuristic data runs. These sets of data were then used to produce a series of comparative metrics between POCO-MOEA and exhaustive search.

6.2 Algorithm Lessons

This section covers the overall results of application of POCO-MOEA to the CPP within the model framework (such as POCO). The section moves from the specific (behavior of POCO-MOEA) to the general (MOEAs overall).

6.2.1 POCO-MOEA Metric Results.

In the cases of δ_1 and δ_2, it became clear in all networks that N and M had significant impact, although the impact of N on reducing these metric values was more reliable. Increasing N would almost always reduce either metric, for every network, regardless of M value. By comparison, the improvement brought about by M was significant up until some threshold, then did not provide any additional benefit. In almost every case this threshold seemed to be reached at some point between the $M = 10$ and $M = 50$ data points. This is evidenced by neither δ value improving much for any network between $M = 50$ and $M = 100$, regardless of N value. Determining the point at which M ceases to provide any additional benefit may be of some computational value. If an experimenter is able to determine through observation or experimentation the exact point at which increasing M produces no additional benefit, the experimenter can use such a value to determine the optimum M value. It is believed that this optimum M value differs from network to network; further experimentation would need to take place to determine the validity of this hypothesis

The next conclusion drawn from the use of POCO-MOEA are that both N and M have a significant impact on heuristic speed. The b_{rel} values for the Test network in Section 5.1.4.3 show that the heuristic is not overly efficient except in cases where N and M remain small. In such a case, a significant trade-off must be made between the optimality of exhaustive search and the speed of POCO-MOEA. Even with speed

increases ($N = 200$, $M = 10$ performs in a 0.08454 time fraction of exhaustive, on average), the speed with which exhaustive search can be performed (early tests showed times around 1.25 seconds) may negate the speed benefits brought about by the heuristic. In the instances of larger networks, the speed benefits of POCO-MOEA begin to become more prominent. In the case of the Latnet Network, even the most generous N and M values allowed POCO-MOEA to run over 4 times faster than exhaustive search. The smaller values on the other hand ($N = 200$, $M = 10$) produced runs nearly 500 times faster than their exhaustive counterpart. As network sizes become larger (or potentially more detailed) the value of POCO-MOEA becomes ever more clear for finding a quick acceptable resolution to the CPP.

The last metric, hyp_{rel}, shows very similar characteristics to the δ metrics for all four networks as N and M values are adjusted. The simplified algorithm used to determine hyp_{rel} results in some initial confusion as to achieving values greater than one, but this is due to the approximative nature of the algorithm. More precision for the algorithm would likely have resulted in more accurate results that did not provide technically invalid solutions. Much like the δ metrics, increasing N and M both provide benefits, with N providing steady improvements with increasing values while M increased rapidly up until a certain threshold before plateauing. Unlike with the δ metrics however, M did not cease to improve the hyp_{rel} metric for most networks between $M = 50$ and $M = 100$, merely greatly decreased. This also provides additional information to the experimenter; they now have more information to make the decision on whether the extra hypervolume gained by increasing M is worth the extra time consumed.

An interesting note on performance metrics overall was the apparent dependency of the network topology on performance. In some cases (specifically the Roedunet network), even small values for M and N allows the resulting populations to come

within 1% of the optimum solution in the case of the hyp_{rel} metric. In other cases, such as the Latnet network, no increase in N or M seemed to produce a significant improvement in metrics beyond certain values (for example, hyp_{rel} did not improve much beyond 95% once this value is reached). The current conjecture is the topology of the network has an impact on the way these metrics change over time. In the reports for original POCO and POCO-PSA, more focus was given on the behavior of metrics as they applied to an entire range of networks as retrieved from the Internet Topology Zoo [7, 39]. Little research was devoted to the impact of individual topologies on overall results. It is believed further research into this field may yield significant results in the pursuit not just of the CPP, but network design as well.

6.2.2 Hypothesis Results.

The end result of the metrics highlight our original hypotheses at the beginning of this thesis, which were a) nearly optimal solution sets produced 10 times faster than exhaustive searh, and b) input parameters have a significant impact on the performance of POCO-MOEA. The first proved to be true when N/M values are small and/or the network topology is large. The concept of a 'nearly' optimal network can change depending on who is managing the network; it can be up to a network administrator to determine whether certain values are 'good enough' compared to exhaustive search. In the case of large networks however, determining closeness to optimum may not be possible. The second hypothesis proved to be true for all parameter changes; regardless of the network, increasing N or M increased the execution time for POCO-MOEA. Increasing either metric also improved the results up to certain thresholds for each value. Again, real use of this algorithm would require the determination of a user to determine the best mix of N, M, other input values balanced with execution time to determine the best use of POCO-MOEA.

6.3 Future Work

There are several areas in which future work can be performed on using MOEAs to solve the controller placement problem, at multiple levels of scope. These future work recommendations are discussed below.

6.4 Usage of MOEAs

One of the defining characteristics of MOEAs is the countably infinite number of variations that exist for how to compute the solutions. The Genetic and Evolutionary Computation Conference (GECCO) is an annual meeting dedicated to the latest advancements in this type of computation strategy [47]. Memetic algorithms, evolutionary machine learning, Ant Colony Optimization (ACO) and swarm intelligence are all variations on the Evolutionary Algorithm (EA) paradigm that has shown varied levels of effectiveness depending on the problem being solved. ACO has proven particularly effective at solving the Traveling Salesman Problem (TSP), a problem which has a node layout very similar to those of a 2 dimensional computer network.

6.4.1 Within POCO-MOEA.

There is a large amount of additional information which can provide more depth into the characteristics of how POCO-MOEA performs relative to exhaustive search. For example, the underlying cause of the thresholds for the δ_1, δ_2, and hyp_{rel} values for certain N and M values on certain networks deserves more research. One theory on the cause for this phenomenon involves the relative size of the pareto front generated by POCO-MOEA compared to that created by exhaustive search. In preliminary test runs of POCO-MOEA on the Test network, a population size of $N = 200$ produced a pareto front around one-third the size of the exhaustive pareto front. As N increased, so too did the size of the POCO-MOEA pareto front. Searching for whether a corre-

lation between the sizes of the pareto fronts and the thresholds of certain metrics may yield more information about the effects of searching a small fraction of the available search space. In addition, plots showing the trade-offs between the different metrics measured for POCO-MOEA could help an observer determine the 'optimum' settings of N and M values.

The Internet Topology Zoo [10] has over 250 networks available for performance testing. As only four networks are tested for this network, further experimentation may reveal more insight on network dependent POCO-MOEA performance. Specifically, there are another of larger networks which take more time to produce a full set of 30 data runs than is available for this thesis. For example, a network model called TataNld consists of 735 nodes and takes approximately 6 hours to produce an exhaustive set of data. Additionally, the full solution set with a value of $k = 5$ consists of over 1.7e+12 data points and takes over 8 GB of space. Another point of research related to networks would be analysis of how the characteristics of network topology impacts the performance of a POCO-MOEA (or any MOEA). As shown in Chapter V, a network with more centralized nodes in its topology (Roedunet) has worse metrics at $M = 1$ compared to a more diverse topology (Missouri), but converges more closely to optimum values with increased generations.

6.4.2 MOEAs In General.

The POCO-MOEA used for this experiment is only one variation on the MOEA format; there are a wide variety of ways with which to implement the MOEA paradigm. Implementing a version of ACO for the controller placement problem may prove more effective than the POCO-MOEA used for our experiment. Even within a specific MOEA, there are multiple ways to adjust the parameters used for experimentation. For example, in POCO-MOEA, each of the parameters used remain static. The p_c

value in particular remains relatively high. This is a desirable trait in the early generations of the algorithm when the actual extent of all possible solutions along a pareto front are still unknown. However, this same trait becomes less desirable in later generations when good solutions can be accidentally removed. One way to mitigate this potential loss is to reduce the p_c value over time, so that the solutions discovered are more likely to remain, and changes to the solution set are minor. This is similar to the mechanic used with ρ to reduce the T value in the PSA heuristic [7].

6.4.3 The CPP Environment.

This POCO model for determining the usage of MOEAs on a computer network is just that: a model. Ideally, the use of real networks would be preferred to find more accurate methods based on real-time data being pulled from switches in real time. The POCO-PLC version of the POCO program [36] demonstrates the use of the exhaustive algorithm to adjust assignment of controllers based on several of the most common metrics, including Average node-to-controller latency (based on round trip ping times) and controller assignment imbalance. This is done based on live statistics of the PlanetLab academic network, designed for performing distributed network tests.

The POCO-PLC experiment is performed on a small subset of the PlanetLab network, due to the amount of time required to retrieve network statistics and determine optimal controller placements. The application of a heuristic such as POCO-MOEA would allow the use of the PLC program on a much larger subset of the PlanetLab network, possibly the entire network. Future work on the use of MOEAs should include running algorithms on live networks such as PlanetLab or GENI [41].

Even if performing tests on an actual live network are not easily feasible, there are still methods of performing tests on a system that better imitates the behavior of

a real network. Creating networks under the use of simulators like Mininet [23] allow for the specific designation of devices as hosts, controllers, switches, etc. Likewise, Mininet allows for the changing of characteristics of these devices to behave more like their real network counterparts. Performing network performance tests using these tools can provide a more realistic assessment of solutions for the CPP.

Appendix A. POCO File Structure

1.1 Overview

This section describes the overall file structure and program flow of POCO before and after the addition of GA files.

1.2 Common Files

The default framework contains a number of .m files; the following is a sample of the files and their functions:

- poco_GUI - the core file. Responsible for initiating and redrawing the GUI; it is also the starter point for all controller placement evaluations
- allToAllShortestPathMatrix - generates a matrix of the shortest distance between any two nodes in the entire network
- allToSomeShortestPathMatrix - generates a matrix of the shortest distance between two nodes within a subset of the entire network
- calculateMetrics - calculates a variety of metrics for a given selection of k controllers out of n nodes on a network, with the option of an array of weights for each node; called by evaluateSingleInstance. Metrics include:
  - Average Node-to-Controller latency
  - Maximum Node-to-Controller latency
  - Count of controllerless nodes
  - Controller imbalance - difference between maximum and minimum nodes assigned to a controller
  - Average Controller-to-Controller latency
  - Maximum Controller-to-Controller latency
- darken - changes GUI aesthetics

- distFrom - node distance calculator; used by poco_GUI to provide edge distances between nodes
- evaluateControllerFailure - starts evaluations of controller placements for failure free and up to $k-1$ controller failures
- evaluateNodeFailure - starts evaluations of controller placements for failure free and up on 2 node failures
- evaluatePlacements - evaluates metrics, unused
- evaluatePlacementsFast - same as evaluatePlacements, but does not perform controller-to-controller latency calculations; used by the PLC calculators
- evaluateSingleInstance - decides on controller placements based on the provided distanceMatrix (see below); used by every other evaluation program to calculate metrics in a failure free scenario
- findFullCoveragePlacements - used for finding a minimum k controller count to provide full resilient coverage
- importGephiGraphML - imports GephiGraphML .xml files [48]
- importGraphML - imports GraphML files
- importSNDlibXML - imports SNDlibXML files
- loadNewPLCfile - imports PlanetLab topology file based on provide .mat and .csv file
- mercatorProjection - unknown at this time
- paretobest2fastlogic - builds the pareto optimal set of solutions based on two vector inputs (sets of solutions)
- paretobest4fastlogic - builds the pareto optimal set of solutions based on four vectors inputs (sets of solutions)
- plotBarsPLC - plots topologies based on metrics from a PlanetLab input
- plotPareto - plots the solution space for two different metrics

- plotParetoPLC - plots the solution space for ongoing Planet Lab inputs
- plotTopology - plots the network topology for a static network
- plotTopologyPLC - plots the live network topology given statistics from a PlanetLab input

Appendix B. Preliminary Data Runs

This appendix shows some of the older non-factorial data runs that were performed while learning the initial characteristics of how the POCO-MOEA heuristic performed.

2.1 Test Run Format

The experiments were performed on the same networks that appear in Chapter V

2.1.1 Test Network.

The box plot of the δ_1 metric values for the Test Network are shown in Figure 32. The first four boxplots show steady improvement in the average closeness of the solutions generated by POCO-MOEA in comparison to those generated by the exhaustive solution as the number of generations increases from 1 to 40. However, at this point it appears that further increasing the number of generations run provides little improvement, with the fifth boxplot of 400 generations actually showing a decrease in solution quality. Furthermore, plots 6 and 7 demonstrate that an increase in the number of solutions allows for an even closer average approachment to the optimum pareto front. Meanwhile, the 8th boxplot shows that a decrease in the crossover value can cause a wider variety of solutions in the population when all other values are kept the same (see plot 3).

The box plot of the δ_2 metric exposes a pattern very similar to that of δ_1. Plots 1 through 4 show a steady improvement in the worst case closeness of a heuristic solution to it's closest optimal point. One difference between the $delta_1$ and $delta_2$ plots shows in the 5th plot, where the large number of generations causes the $delta_2$ value to nearly collapse to a single point. Again, plots 6 and 7 demonstrate that increasing the size of the population allows for an even greater improvement on approaching the optimal

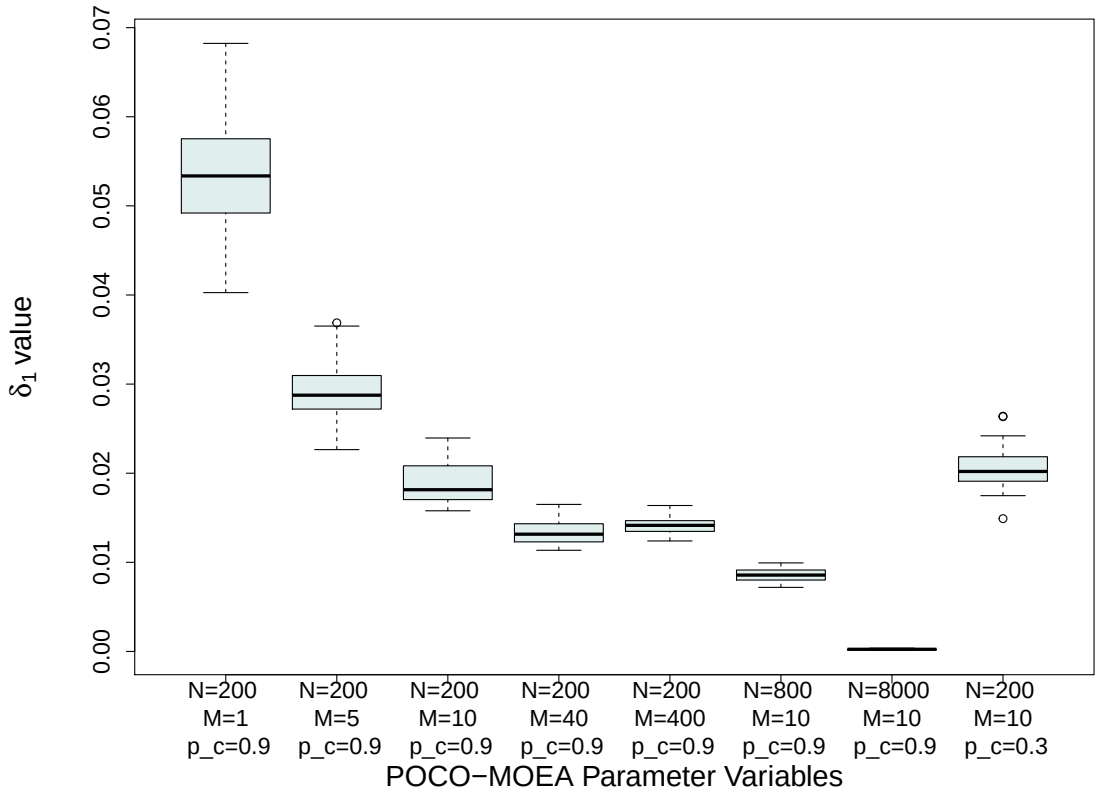

Figure 32. Test Network δ_1 Heuristic Parameter Comparison

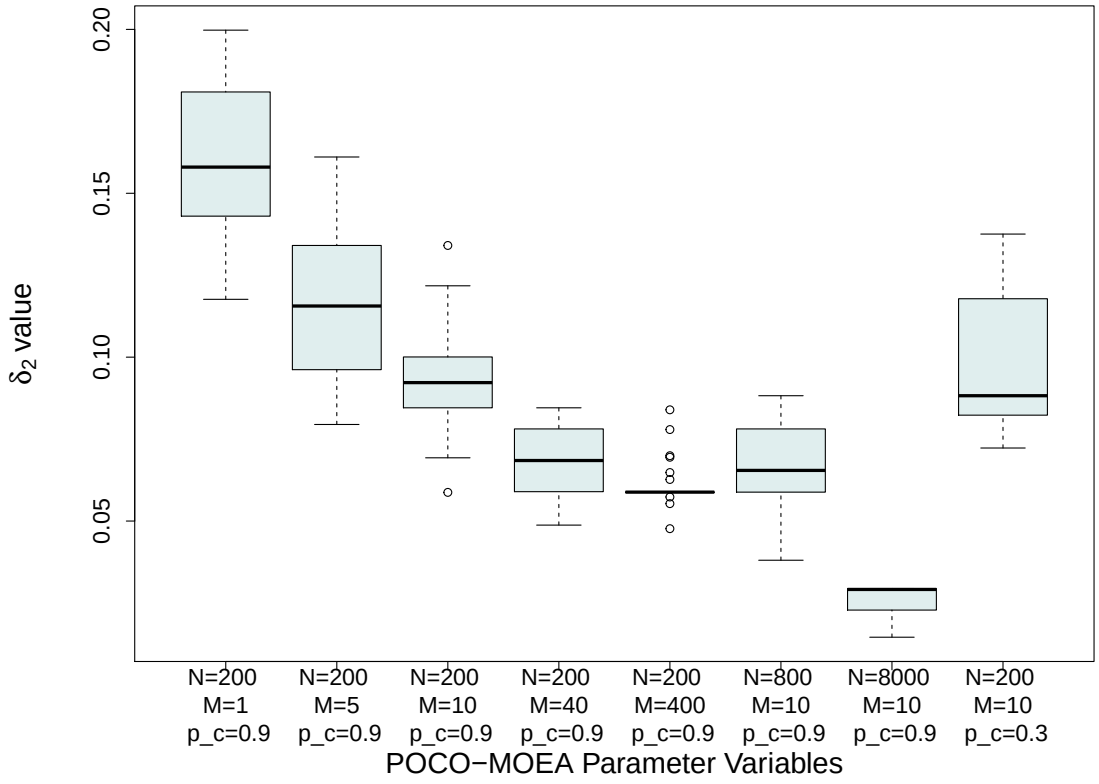

Figure 33. Test Network δ_2 Heuristic Parameter Comparison

solution compared to merely increasing the number of generations. The last plot, with the lower crossover probability demonstrates an on average greater variety of δ_2 values when compared to plot 3.

The box plots for the calculated hypervolumes at each parameter are given in Figure 34. These plots show the fraction of hypervolume values between the heuristic and the exhaustive tests. A hyp_{rel} value closer to 1 indicates a heuristic that comes closer to the optimum hypervolume. As Figure 34 shows, a steady increase in hyp_{rel} values comes with a steady increase in the number of generations run, even between the boxplots of 40 generations and 400 generations. Likewise, increasing the number of solutions in the population can also improve the hyp_{rel} value, to the point where a sufficiently high population (plot 7) can achieve near optimal hypervolume values.

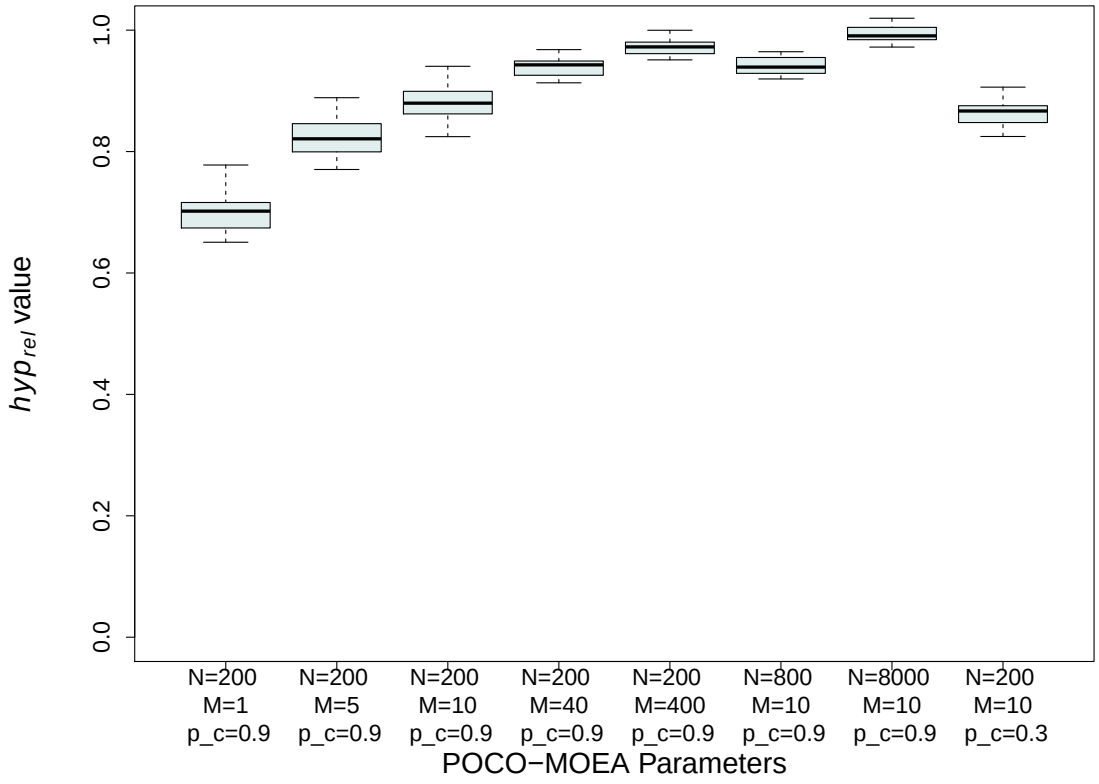

Figure 34. Test Network hyp_{rel} Parameter Comparison

The last plot indicates that a lower p_c value results in a lower hyp_{rel} value on average (compared to plot 3). Conducting a t-test between the results of these two plots gives a p-value of 0.03223, indicating that these two values are indeed different at the 95% confidence level.

2.1.2 b_{rel} and Hypervolume Values.

The b_rel boxplots are shown in Figure 35. The horizontal dashed line in the figure represents a b_{rel} value of 1, or the time taken by the exhaustive search. In this specific network, heuristics with a large number of generations (plot 5), or a large population size (plot 7) actually take longer than the exhaustive search. In such a case, the loss in optimality may not be worth the time saved.

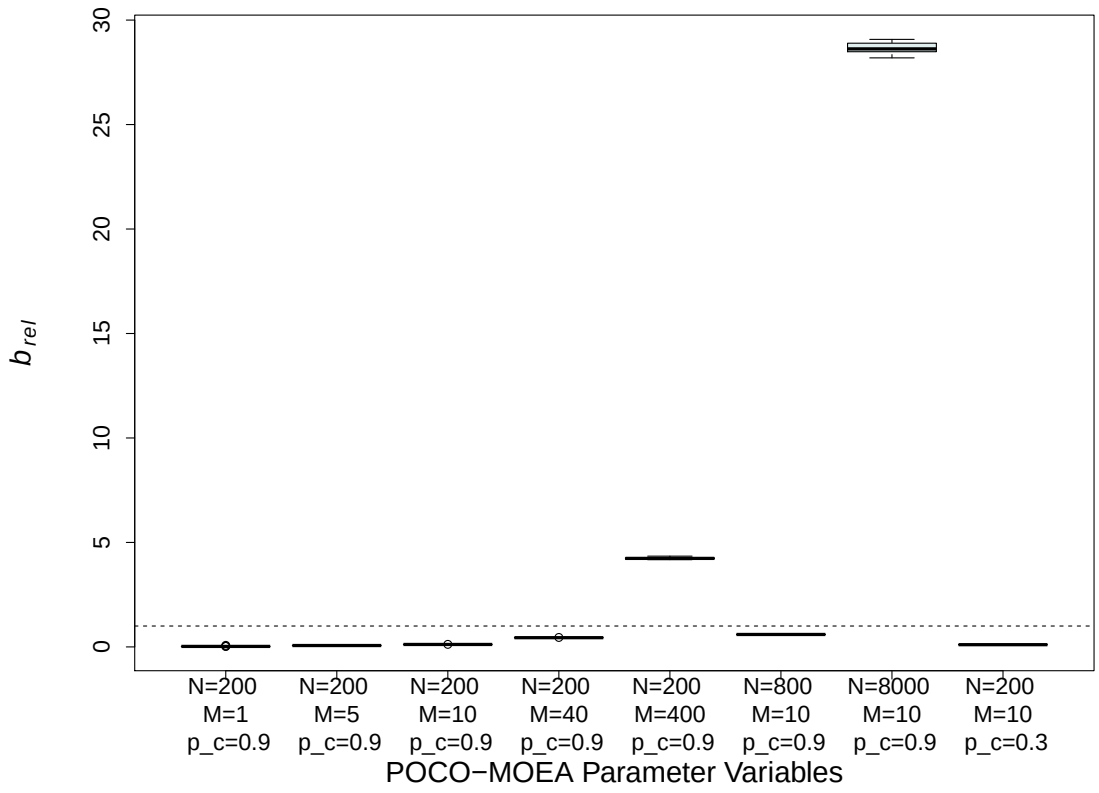

Figure 35. Test Network b_{rel} Parameter Comparison

Bibliography

1. Theophilus Benson, Aditya Akella, and David A Maltz. Unraveling the Complexity of Network Management. In *NSDI*, pages 335–348, 2009.

2. George Vanecek. GeoPlex: Universal Service Platform for IP Network-Based Services, Retrieved 26 May 2015, https://www.cerias.purdue.edu/news_and_events/events/security_seminar/details/index/56218-nozckzkv1f3c-372-hq15ntbspk31bq8w, October 1997.

3. Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A Clean Slate 4D Approach to Network Control and Management. *SIGCOMM Computer Communication Review*, 35(5):41–54, October 2005.

4. Brandon Heller, Rob Sherwood, and Nick McKeown. The Controller Placement Problem. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 7–12, New York, NY, USA, 2012. ACM.

5. Gérard Cornuéjols, George L Nemhauser, and Lairemce A Wolsey. The Uncapacitated Facility Location Problem. Technical report, Carnegie-Mellon University Management Sciences Research Group, August 1983.

6. Carlos Coello Coello, Gary B Lamont, and David A Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer Science & Business Media, 2nd edition, 2007.

7. D. Hock, S. Gebert, M. Hartmann, T. Zinner, and P. Tran-Gia. POCO-Framework for Pareto-Optimal Resilient Controller Placement in SDN-Based Core Networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–2, May 2014.

8. D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia. Pareto-Optimal Resilient Controller Placement in SDN-Based Core Networks. In *Teletraffic Congress (ITC), 2013 25th International*, pages 1–9, September 2013.

9. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, April 2002.

10. S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet Topology Zoo. *Selected Areas in Communications, IEEE Journal on*, 29(9):1765–1775, October 2011.

11. D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: A Comprehensive

Survey. *Proceedings of the IEEE*, 103(1):14–76, January 2015.

12. Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A Distributed Control Platform for Large-Scale Production Networks. In *In Proceedings OSDI*, 2010.

13. Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking Control of the Enterprise. *SIGCOMM Computer Communication Review*, 37(4):1–12, August 2007.

14. Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

15. Floodlight is an Open SDN Controller, Retrieved 31 May 2015, http://www.projectfloodlight.org/floodlight/.

16. OpenDaylight - An Open Source Community and Meritocracy for Software- Defined Networking, Retrieved 31 May 2015, http://www.opendaylight.org/publications/opendaylight-open-source-community-and-meritocracy-software-defined-networking.

17. Network Functions Virtualisation (NFV): Architectural Framework, Retrieved 1 October 2015, http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf, 2013.

18. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Computer Communication Review*, 38(2):69–74, March 2008.

19. OpenFlow Switch Specification: Version 1.0.0 (Wire Protocol 0x01), Retrieved 3 September 2015, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf, December 2009.

20. OpenFlow Switch Specification: Version 1.3.0 (Wire Protocol 0x04), Retrieved 3 September 2015, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf, June 2012.

21. OpenFlow Switch Specification: Version 1.5.0 (Protocol Version 0x06), Retrieved 3 September 2015, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.pdf, December 2014.

22. OpenFlow Switch Specification: Version 1.5.1 (Protocol Version 0x06), Retrieved 3 September 2015, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf, March 2015.

23. Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

24. Eclipse Public License - v 1.0, Retrieved 3 September 2015, https://www.eclipse.org/legal/epl-v10.html.

25. S.H. Yeganeh, A. Tootoonchian, and Y. Ganjali. On Scalability of Software-Defined Networking. *Communications Magazine, IEEE*, 51(2):136–141, February 2013.

26. Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 19–24, New York, NY, USA, 2012. ACM.

27. Yan-Nan HU, Wen-Dong WANG, Xiang-Yang GONG, Xi-Rong QUE, and Shi-Duan CHENG. On the Placement of Controllers in Software-Defined Networks. *The Journal of China Universities of Posts and Telecommunications*, 19, Supplement 2(0):92 – 171, 2012.

28. Zhuolin Jiang, Zhe Lin, and L.S. Davis. Learning a Discriminative Dictionary for Sparse Coding via Label Consistent K-SVD. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1697–1704, June 2011.

29. Minzhe Guo and P. Bhattacharya. Controller Placement for Improving Resilience of Software-Defined Networks. In *Networking and Distributed Computing (ICNDC), 2013 Fourth International Conference on*, pages 23–27, December 2013.

30. Y. Jiménez, C. Cervello-Pastor, and A.J. Garcia. On the Controller Placement for Designing a Distributed SDN Control Layer. In *Networking Conference, 2014 IFIP*, pages 1–9, June 2014.

31. Mathieu Bastian, Sebasien Heymann, Mathieu Jacomy, et al. Gephi: An Open Source Software for Exploring and Manipulating Networks. *ICWSM*, 8:361–362, 2009.

32. Sébastien Auroux and Holger Karl. Flow Processing-Aware Controller Placement in Wireless DenseNets. In *Proceedings of the 25th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–6,

2014.

33. A. Sallahi and M. St-Hilaire. Optimal Model for the Controller Placement Problem in Software Defined Networks. *Communications Letters, IEEE*, 19(1):30–33, January 2015.

34. D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou. Adaptive Resource Management and Control in Software Defined Networks. *Network and Service Management, IEEE Transactions on*, 12(1):18–33, March 2015.

35. David A Van Veldhuizen and Gary B Lamont. Evolutionary Computation and Convergence to a Pareto Front. In *Late breaking papers at the genetic programming 1998 conference*, pages 221–228. Citeseer, 1998.

36. D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tran-Gia. POCO-PLC: Enabling Dynamic Pareto-Optimal Resilient Controller Placement in SDN Networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 115–116, April 2014.

37. Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *SIGCOMM Computer Communications Review*, 33(3):3–12, July 2003.

38. PlanetLab, Retrieved 30 October 2015, https://www.planet-lab.org/.

39. S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann. Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks. *Network and Service Management, IEEE Transactions on*, 12(1):4–17, March 2015.

40. S. Lange, S. Gebert, J. Spoerhase, P. Rygielski, T. Zinner, S. Kounev, and Phuoc Tran-Gia. Specialized Heuristics for the Controller Placement Problem in Large Scale SDN Networks. In *Teletraffic Congress (ITC 27), 2015 27th International*, pages 210–218, September 2015.

41. Mark Berman, Jeffrey S. Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. GENI: A Federated Testbed for Innovative Network Experiments. *Computer Networks*, 61:5 – 23, 2014. Special Issue on Future Internet Testbeds: Part I.

42. Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. Many-Objective Evolutionary Algorithms: A Survey. *ACM Comput. Surv.*, 48(1):13:1–13:35, September 2015.

43. Pareto Front by Yi Cao, Retrieved 4 December 2015, http://www.mathworks.com/matlabcentral/fileexchange/17251-pareto-front.

44. Hypervolume Indicator by Yi Cao, Retrieved 29 December 2015, http://www.mathworks.com/matlabcentral/fileexchange/19651-hypervolume-indicator.

45. Piotr Czyżżak and Adrezej Jaszkiewicz. Pareto Simulated Annealing: A Metaheuristic Technique for Multiple-Objective Combinatorial Optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47, 1998.

46. Package 'h5', Retrieved 21 December 2015, https://cran.r-project.org/web/packages/h5/h5.pdf.

47. Genetic and Evolutionary Computation Conference, Retrieved 21 December 2015, http://gecco-2016.sigevo.org/index.html/homepage.

48. Gephi: The Open Graph Viz Platform, Retrieved 31 August 2015, https://gephi.github.io/.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24-03-2016	Master's Thesis	May 2014 — Mar 2016

4. TITLE AND SUBTITLE

POCO-MOEA: Using Evolutionary Algorithms to Solve the Controller Placement Problem

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Harned, Scott, I., Capt, USAF

5d. PROJECT NUMBER

16G236

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-16-M-021

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

INTENTIONALLY LEFT BLANK

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

One of the central tenets of a Software Defined Network (SDN) is the use of controllers, which are responsible for managing how traffic flows through switches, routers, and other data passing devices on a computer network. Most modern SDNs use multiple controllers to divide responsibility for network switches while keeping communication latency low. A problem that has emerged since approximately 2011 is the decision of where to place these controllers to create the most 'optimum' network. This is known as the Controller Placement Problem (CPP). Such a decision is subject to multiple and sometimes conflicting goals, making the CPP a type of Multi-Objective Problem (MOP). The theory of this thesis is that an MOEA can produce solutions to the CPP which are 'nearly optimal' while keeping execution time low compared to an exhaustive 'optimal' search. This research extends a network modeling tool called the Pareto Optimal Controller Placement (POCO) Framework with custom designed MOEA, called POCO-MOEA.

15. SUBJECT TERMS

Thesis, Algorithm, Software Defined Network, Controller Placement Problem, SDN, MOP, POCO, MOEA, PSA

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. B. Mullins, AFIT/ENG
U	U	U	U	132	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4555; barry.mullins@afit.edu